Scalability in Distributed Multimedia Systems

Mari Korkea-aho
Helsinki University of Technology
Laboratory of Information Processing Science

Master's thesis

November 5, 1995

Acknowledgements

The work in this thesis[1] is a part of the OtaOnline research project at the Helsinki University of Technology. I would like to take the opportunity to thank all those who have contributed to the work.

First, I would like to express my gratitude to my supervisor Prof. Shosta Sulonen for all his support, guidance, constructive criticism and ideas. I would also like to thank my advisor Antti Ylä-Jääski at the Nokia Research Centre, as well as Esa Turtiainen at the Helsinki University of Technology for their guidance and support.

I am grateful to my colleagues Tuomas Puskala, Marko Turpeinen and Janne Saarela for all their help and comments. I would also like to thank Heikki Hämmäinen, Jaripekka Salminen and Sami Tikka at the Nokia Research Centre for inspiring discussions.

I would like to thank Nokia for providing financial support for the work.

I also owe my gratitude to my parents for their encouragement and support, and to my brother and sister for making life so enjoyable.

Finally, I would like to thank my friend Thomas for his immense patience and support, without which this work might never have been completed.

Mari Korkea-aho
November 5, 1995

[1]The thesis is available on the Internet, at URL http://www.hut.fi/~mkorkeaa/thesis.html

Contents

1 Introduction

In recent years the interest in and the use of *distributed multimedia systems* have increased rapidly. One reason for this is the wide implementation potential that such systems offer. Another reason is the flexibility that they offer to the user. The user can access remote services, freely choose what to receive and personally control the manner and time of receipt of the data.

The progress of distributed multimedia systems has been feasible, because of advances in network and computer technology. There are, however, still many technical challenges, since multimedia data differs significantly from traditional data that current computer systems are designed to handle.

Scalability is one of the most important factors when designing distributed multimedia systems which provide interactive services for a wide clientele. To maintain the clients the system must work fast and reliably all the time. To do this the system must be able to adapt itself to different numbers of users and amounts of data, without resource problems or performance bottlenecks. Scalability might be the crucial factor for the success or failure of a service.

To be able to scale a distributed multimedia system its structure and purpose must be thoroughly understood. The system architecture, and the characteristics of the provided service and data affect what scaling methods should be used.

1.1 Objectives and scope of the study

The aim of this study is to understand what causes scalability problems, how a distributed multimedia system can be scaled and how different scaling methods suit different types of multimedia data and system architectures. Based on the review and analysis of different scaling methods a scalable multimedia server architecture will be proposed.

The study of existing and proposed scaling methods will concentrate on the distributed information system World Wide Web. This since the World Wide Web is currently the most widely used distributed multimedia system and the work done in the OtaOnline project is based on it. However, this should not be a problem, since the suggested methods can also be implemented and used in other system environments.

Most of the existing scaling methods try to distribute the load, by distributing users to several servers. The problem with these schemes is that the server selection is mainly based on system defaults or the choice of the user. This can cause uneven load distribution and obstruct the scalability of a system. A system where users would be directed dynamically to an appropriate server according to location, server load, available network bandwidth or some other system metrics,

might be an improvement to current scaling schemes. Such a scheme called the *super server concept* will be presented and evaluated in this thesis.

The super server concept was implemented in order to determine its efficiency. To simplify comparison to other scaling methods it was constructed as a World Wide Web server, which can be extended to work according to the principles of the super server concept. The system is only built for fair comparison of methods. It is not intended to be a full-scale World Wide Web server.

When considering a scalable server architecture, I have tried to take into account properties that an interactive news system, like the one in the OtaOnline project, might require. These include distribution of the data, personalized versions of the available data, dynamic creation of the presentation and gathering and processing of system information. The details of the implementation of these schemes are not considered in this work.

The emphasis in this thesis is on scalability through distribution of user requests. Enhanced scalability through server hardware or software improvements or compression of data will not be considered.

1.2 The OtaOnline project

This work is a part of the OtaOnline project at the Helsinki University of Technology. The goal of the OtaOnline project is to develop an interactive news system, a so-called electronic newspaper. The system should be scalable and able to offer personalized multimedia news. In addition to the development of a scalable system architecture, the research areas of the project cover production systems for multimedia services, structured modeling of multimedia products, and personalization of multimedia services.

To be able to experiment with the possibilities of a network media and electronic publishing, OtaOnline has been implemented as a World Wide Web service at the campus of the Helsinki University of Technology.

1.3 Organization of this document

To start with, multimedia is defined in Chapter 2. In Chapter 3, distributed multimedia systems and possible applications are presented. Especially, the architecture of distributed multimedia systems and the demands on the different system components are considered.

In Chapter 4, the scalability of a multimedia system is discussed. First, scalability is defined. Then the importance of scalability in a distributed multimedia system and factors causing scaling problems are reviewed. After this, it is taken

into consideration how users experience the scalability of the system. Finally, different methods to get the system to scale better are explored. To be able to understand the impact of the different methods, characteristics of multimedia data and services are first considered. After this, the methods are presented and compared.

Chapter 5 introduces the World Wide Web as an example of a distributed multimedia system. The idea and functionality of the World Wide Web are first presented, whereafter the performance of the used HTTP protocol and World Wide Web servers are examined. In Chapter 6, the current scaling methods of the World Wide Web are reviewed, along with related research and planned implementations.

In Chapter 7, the proposed dynamic scaling method, the so-called *super server concept*, is discussed. The possible ways of implementation are presented and evaluated. In Chapter 8, the super server concept is tested. Based on the analysis in the previous chapters, a scalable server architecture and recommendations for future work is presented in Chapter 9.

In Chapter 10, the work in this thesis is summarized.

2 Multimedia

Multimedia is defined as the composition and simultaneous use of data in different media forms: text, voice, images, animation, graphics, audio, video, etc. [Koegel Buford] [Gibbs and Tsichritzis]. The use of multimedia involves incorporating the modes of information naturally used by humans into computing.

During the last years multimedia has evolved to the fastest progressing, most popular, and revolutionary area of computing. The reason for this rapid development is the tremendous potential of the technology. Multimedia enables a natural and simple interaction between man and machine. With its help it is easy to present and access information. The industry and academic community have realized the potential and are developing multimedia applications, such as news-on-demand, home shopping and video-on-demand. The development of such multimedia applications will lead to the merging of computing, communication and broadcasting industries. Some even state that multimedia is poised to change the life style of society.

The development of multimedia systems raises many technical challenges. This is because multimedia data differs significantly from traditional text data that conventional computer systems are built to handle. The differences are [Jadav and Choudhary]:

- **Multiple data streams**: A multimedia object can consist of text, audio, video and image data. These data types have very different storage space and retrieval rate requirements. The design choices include storing data of the same type together, or storing data which belong to the same object together. In either case, multimedia data adds a whole new dimension to the mechanism used to store, retrieve and manipulate data.

- **Real-time retrieval requirements**: Video and audio data are characterized by the fact that they must be presented to the user, and hence retrieved and transported, in real time. In addition, compound objects usually require two or more data types to be synchronized as the object is played out.

- **Large data size**: Typical video or audio objects are much larger than typical text objects. For example a one-and-half-hour MPEG-1 format movie requires about 1 gigabytes of storage. Hence, retrieval and transportation mechanisms must be fast, and they should also provide large storage capacity and transfer bandwidth, respectively. See Table 1 for storage and bandwidth requirements for different data types.

These features of multimedia data strain the capacities of current technology.

Consequently, to make multimedia applications feasible, new approaches and techniques need to be developed for storage, processing and communication.

Data type	Object type	Size and bandwidth	Storage requirements (1 GB can store)
Text	ASCII	2 kB/page	524 288 pages
Image	Low-resolution uncompressed bitmap	64 kB/image	16 384 images
Image	High-resolution compressed bitmap, compression rate 15:1	4MB/image	256 images
Audio	Phone quality 8 kHz 8 bits (mono)	62.5 kb/s	37.2 hours
Audio	CD quality 44.1 kHz 16 bits	1.35 Mb/s	1.6 hours
Video	Digital MPEG-1 352x240x24 bits 30 frames/second	1.5 Mb/s	1.5 hours
Video	Digital MPEG-2 720x480x24 bits 30 frames/second	6 Mb/s	22 min

Table 1: Bandwidth and storage requirements for different data types

3 Distributed multimedia systems

Research and development efforts in multimedia fall into two groups. One group concentrates on *stand-alone multimedia workstations* and associated software systems and tools. The other combines multimedia with distributed systems [Fuhrt]. The *distributed multimedia system* approach offers a broader spectrum of implementation possibilities in comparison to stand-alone systems. But in addition to the possibilities they also add a new dimension to the system complexity.

A distributed multimedia system combines a variety of multimedia information resources connected over a network into an application used by the client. In this manner, the user can access different remote information sources and services. The system is based on interaction between the user and the application. The user can decide when to receive what data and can also control the data flow.

This is a radical shift from conventional broadcast service. In such services, typified by cable television, clients can neither control the programs they view

nor schedule the viewing time of programs to suit their preferences. The user is also flooded with irrelevant information, without a possibility to choose only the information of interest.

The flexibility multimedia systems provide for the user has created many potential applications and services. Some applications are presented in Table 2.

Application	Description
Interactive multimedia news	Multimedia news tailored according to customer interests. The customer can get more details on selected stories. Interactive selection and retrieval.
Video-on-demand	Customers can select and play movies with full VCR capabilities.
Interactive multiparticipant games	Customers can play distributed virtual reality games.
Home shopping	Customers can browse product catalogs, select and purchase products.
Distant learning	The student can take part in courses taught at remote sites. The student can tailor the course to individual preferences and time constraint.
Multimedia mailing systems	Electronic messages can contain audio, video, images, and other media.
Multimedia conferencing	Participants share a virtual workspace where they can exchange information and opinions via multimedia data.

Table 2: Different distributed multimedia system applications

The applications mentioned in Table 2 already exist in the Internet and networks of commercial providers. To a high extent the existing applications are still quite simple because of performance restrictions. They mostly use text and image data and only include a limited amount of video and audio data of low quality. This is because of limited network and processing resources.

Because of the high potential of multimedia systems, applications are evolving quite fast under continuous work. There are many development and research efforts world-wide. An example is the OtaOnline project at the Helsinki University of Technology where we are developing an interactive multimedia news system.

Many distributed multimedia applications are not feasible on large scale with current technology. Many challenging problems remain to be investigated and resolved for the further growth of multimedia systems. Among these high-speed networks, large-capacity storage devices, fast processors, multimedia data management, data compression algorithms and architectures of distributed systems.

3.1 Architecture

A distributed multimedia system consists of three basic components (see Figure 1). The user interface or the terminal deals with the issues related to presentation and manipulation of multimedia objects and the interaction with the user. The network provides the communication mechanism between the user and the server. The server is responsible for managing multimedia databases and composing general multimedia objects for the users. The composition of an object is a complex process of integrating and synchronizing multimedia data for transport, display and manipulation. The system usually consists of multiple users, servers and networks [Berra et al.].

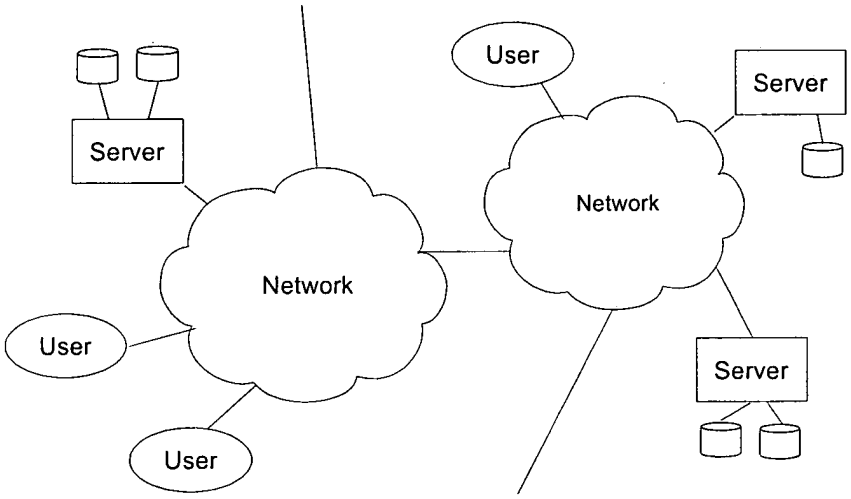

Figure 1: Architecture of a distributed multimedia system

3.1.1 User terminal

A multimedia terminal consists of a computer with special hardware such as a microphone, high-resolution graphics display, stereo speakers and a network interface. The user interacts with the system via a computer keyboard, mouse or a hand-held remote control.

Many of the current user terminals still resemble traditional computers. Because of this, additional development work is still required before the terminals can meet the requirements of multimedia data and the user.

Because of the large size of multimedia objects and real-time requirements the multimedia terminal or the network adapter should include large data buffers. To restore the temporal relationship of a data stream, stream handlers should be connected to the data buffers. To synchronize the possible multiple data streams

and to control the stream handlers, a synchronization and streaming manager is required. Since multimedia data objects are large, the terminal should also include compression and decompression hardware.

3.1.2 Network

Multimedia communication differs from traditional communication (see Table 3). The multimedia traffic requires transfer of large volumes of data at very high speeds, even when the data is compressed. Especially for interactive multimedia communication the network must provide low latency. Continuous media, as video and audio, require guarantees of minimum bandwidth and maximum end-to-end delay. The variation in delay, referred to as jitter, and loss of data must also be bound.

Traditional networks are used to provide error-free transmission. However, most multimedia applications can tolerate some errors in transmission due to corruption or packet loss without retransmission or correction. In some cases, to meet real-time delivery requirements or to achieve synchronization, some packets are even discarded [Fuhrt].

Characteristics	Data transfer	Multimedia transfer
Data rate	Low	High
Traffic pattern	Bursty	Stream-oriented highly bursty
Reliability requirements	No loss	Some loss
Latency time requirements	None	Low (for example, 20 ms)
Mode of communication	Point-to-point	Multipoint
Temporal relationship	None	Synchronized transmission

Table 3: Traditional communication versus multimedia communication [Fuhrt]

The increasing popularity of the Internet and the fact that the infrastructure already exists suggests that the Internet could be used for distributed multimedia systems. It is, however, highly inadequate to handle the high volume of multimedia traffic. The telephone and common antenna TV (CATV) cable networks are also a possibility to support interactive multimedia at user homes due to their wide deployment [Little and Venkatesh].

Traditional networks do not suit multimedia communication. Transmission characteristics of existing Ethernet and Internet protocols (CSMA/CD, TCP/IP) do not support the high-bandwidth, low-latency requirements of audio- and video-based applications.

Ethernet provides only a bandwidth of 10 Mb/s. This is inadequate for most multimedia applications. Moreover, its access time is not bound, and its latency

and jitter are unpredictable. New protocols which are considered for carrying multimedia data include the 100 Mb/s Ethernet standard, Distributed Queue Dual Bus (DQDB), Fiber Distributed Data Interface (FDDI) and Asynchronous Transfer Mode (ATM). The first three have bandwidths of the order of 100 Mb/s. ATM enables a bandwidth of 155-622 Mb/s, depending on the characteristics of the network.

FDDI has in its synchronized mode low access latency and low jitter. FDDI also guarantees a bounded access delay and a predictable average bandwidth for synchronous traffic. Due to its high cost FDDI is at the moment used primarily for backbone networks.

Asynchronous Transfer Mode (ATM) is rapidly emerging as the future protocol for multimedia communication. ATM provides great flexibility in bandwidth allocation by assigning fixed length packets, called cells, to support virtual connections. ATM can also increase the bandwidth efficiency by buffering and statistically multiplexing bursty traffic at the expense of cell delay and loss [Fuhrt]. For the Internet, the Internet Engineering Task Force (IETF) is working on a TCP/IP interface for ATM [Little and Venkatesh].

3.1.3 Multimedia server

Current personal computers, workstations and servers are designed to handle traditional forms of data. Their performance is optimized for a scientific or transaction-oriented type of workload. These systems do not perform well for multimedia data, requiring fast data retrieval and guaranteed real-time capabilities. The I/O capacity is usually a severe bottleneck.

Requirements for multimedia servers [Jadav and Choudhary]:

- **Minimal response time**: A crucial factor for the success of multimedia services is the response time seen by the client. The server must be able to minimize response times to live up to the expectations of the user.
- **Fast processing capability and low data access rates**: To guarantee fast response times, client requests should be processed fast and data access rates should be minimized.
- **Reliability and availability**: Like any other kind of server, a multimedia server must be reliable. The larger the number of users and volume of data is handled by the server, the more difficult it is to guarantee reliability. To provide fault tolerance special hardware and software mechanisms must be employed. Since client requests may arrive at any time, the time the server is unavailable should be minimized.

- **Ability to sustain guaranteed number of streams**: Another important factor is the maximum number of data streams the server can simultaneously handle. This affects the total number of clients the server can serve.

- **Real-time delivery**: To be able to deliver multimedia data, the server should support real-time delivery. This poses profound requirements on the resource scheduling at the operating system level. The server should be able to guarantee real-time delivery for individual streams as well as for all the streams combined together. For this accurate real-time operating systems have to be developed.

- **High storage capacity**: To be able to store multimedia data and a large variety of information the server must have a large storage capacity. To sustain the delivery requirements of multimedia data, the server may be required to compress and encode video and image data prior to transport or storage. The performance of compression and signal processing should be optimized. This might require special hardware.

- **Quality of Service (QoS) requirements**: The Quality of Service (QoS) is a set of parameters describing the tolerable end-to-end delay, throughput, and the level of reliability in multimedia communication and presentation. QoS requirements of clients are an important factor that affects the usage of the server. The server should be able to provide and adapt itself to different QoS requirements, according to the characteristics of the client's terminal, the network connection and the requested data type.

- **Exploit user access patterns**: The server should also be able to trap and exploit dynamic user behavior, minimizing system load and network traffic. For example, by analyzing data access rates and times, popular data could be distributed closer to users in periods of low network load.

- **Ability to handle different types of traffic**: A multimedia server should be able to serve multiple real-time data streams simultaneously, but it must also be able to provide satisfactory service to non-real-time data. It should be able to handle control data encountered when loading new data from other servers or storage repositories, billing and accounting data and communication between intelligent personal agents. Agents are autonomous programs selecting and managing data according to user preferences [Turpeinen].

- **Cost effectiveness**: A very important requirement governing the future of multimedia servers is the cost effectiveness. The server must be affordable.

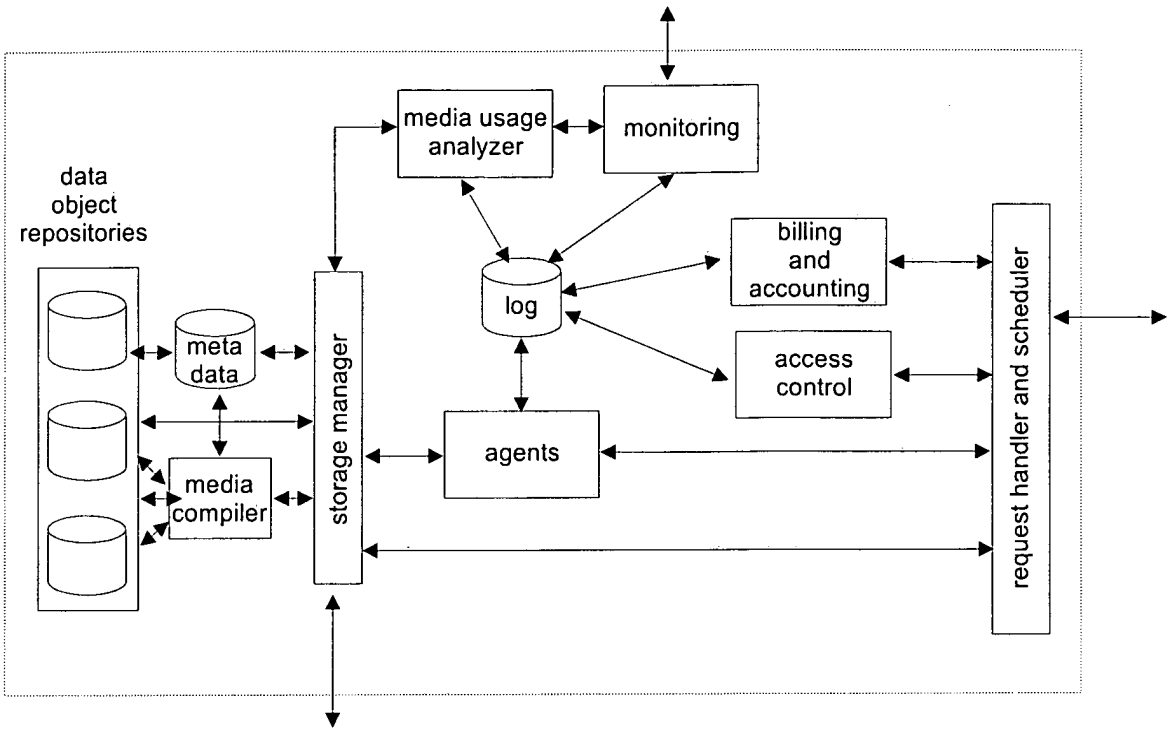

Figure 2: Possible components and their internal relationship in a multimedia server

To satisfy the stringent system requirements the server architecture is quite complex. Figure 2 presents possible components in a multimedia server and their internal relations. The internal structure of the server and needed components depends on the purpose of the server. The presented server architecture is based on consideration of what the OtaOnline interactive news system requires.

The scheduler and request handler takes care of the I/O data flow of the server. It handles the requests and manages the data flow to and from the server.

To be able to restrict the access only to certain users or network domains, the server must include access control. The access control manages which requests are allowed. Since many planned systems will include commercial services a billing and accounting module must be present.

The server requires administration and supervision. For this all transactions are logged in a log repository and a monitoring tool is provided. Since there will be a huge amount of log data available, means for filtering out the interesting information must exist. This could be done by a media usage analyzer which filters out the interesting log information and presents it in an understandable format.

To manage the large amount of data a storage manager is provided. The storage manager manages the data in the object repositories and the data flow. Information about the data, so called metadata, could be saved in a separate repository, facilitating the data management and retrieval. To make it possible to serve the data in different formats, the data could be saved in some specified format and then changed to the wanted format by a media compiler. The media usage analyzer could give feedback of usage patterns to the storage manager, helping to decide what information should be stored where.

To be able to manage the data effectively and to be able to offer personalized services, the system should also include agents. The agents would select what data to present to the user, according to user preferences and access patterns. In this manner each user would get a personalized service.

To decrease server load, the processing of requests could be distributed among multiple servers. Every server would be its own entity, but they could then be managed in a centralized or distributed manner. Such approaches need careful consideration and analysis, because of their complexity.

The design of a multimedia server needs thorough planning to meet the stringent requirements. Designing a high-performance multimedia server that can support multiple, simultaneous, full-motion video streams is still challenging. A great deal of work needs to be done in areas like real-time scheduling, parallel I/O, reliability, scalability, dynamic scheduling and caching techniques for multimedia data.

4 Scalability in distributed multimedia systems

A scalable system is one which continues to work even though some variables in the system vary, usually to a great extent. A solution that works fine for a small problem set may turn out to be impractical for the same problem when scaled up to large size [LaLiberte and Braverman]. To achieve scalability, the impact of varying variables must be minimized.

A concrete example of the definition above is a system where the number of users increases. If the system is scalable, the system should manage this increase without resource problems or performance bottlenecks. The system should withstand the increase without changes in system structure or application algorithms.

In a distributed multimedia system the scalability is extremely important. These systems provide interactive services for a wide clientele. To maintain the clients the system must work fast and reliably all the time, and withstand unexpected changes in the number of users and amount of data. This is especially important in a commercial environment. Without clients, the viability of the business would be low.

When designing interactive multimedia systems, special attention should be paid to the scalability of the system. It might be the crucial factor for the success or failure of the service.

4.1 Factors causing scalability problems

The main factors causing scalability problems in a distributed multimedia systems are [LaLiberte and Braverman]:

- **Growth of the user base.** A growing user base can cause serious scalability problems for servers. Requests from users add to the network traffic in proportion to the number of requests, which is proportional to the number of users. Some protocols even perform worse. This causes significant scalability problems for the network.
- **Size of the data objects.** Particularly, the size of audio and video files strains the network and I/O capacity causing scalability problems.
- **Amount of accessible data.** The increasing amount of accessible data, makes data search, access, and management more difficult. This causes processing problems.
- **Non-uniform request distribution.** The interests of users are not evenly distributed over the day and available servers. This puts strains on the servers and network at certain times of the day or at certain locations.

Personally, I consider the non-uniform request distribution followed by the growth of the user base and the large size of the multimedia objects the most serious problems in current systems. This while the non-uniform request distribution easily causes sudden unexpected load peaks, where as the growth of the user base and amount of data is usually slightly more controlled.

Because of above mentioned factors, the system might run out of resources, such as network, processing, I/O or storage capacity. It is often very expensive to acquire more needed resources, or to have unused resource capacity in advance in the system. This is particularly valid for network capacity. Although there are enough resources available, resource allocation problems can sometimes occure due to the dynamic nature of the system.

In current systems it is the network bandwidth, and the I/O and processor capacity of the server and client that are the main cause of scalability problems.

Multimedia systems should be constructed to be more independent of resource limitations and allocation problems to guarantee scalability.

4.2 How a user experiences scalability

The user experiences the scalability of a system in how fast and accurately the system responds to actions. The response should be received in an acceptable time without any errors. These qualities can be measured via system response time, availability and reliability.

4.2.1 Response time

Response time means the time between the moment when a user gives an input and the moment when a user receives an answer from the system. The response time in a distributed multimedia system consists of all the delays created at the source site, in the network, and at the receiver site (Figure 3).

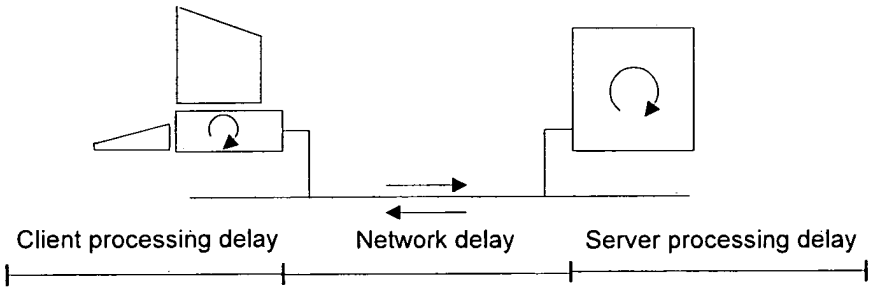

Figure 3: Delay factors in a distributed multimedia system [Hautaniemi]

The possible reasons for the delays and their size depend on the system components and the characteristics of the transport media. Figure 4 tries to demonstrate these possible delay factors. It presents what the delay from the server to the client consists of, in case of real-time video and audio and stored multimedia objects. The response time naturally consists of the delays in both directions.

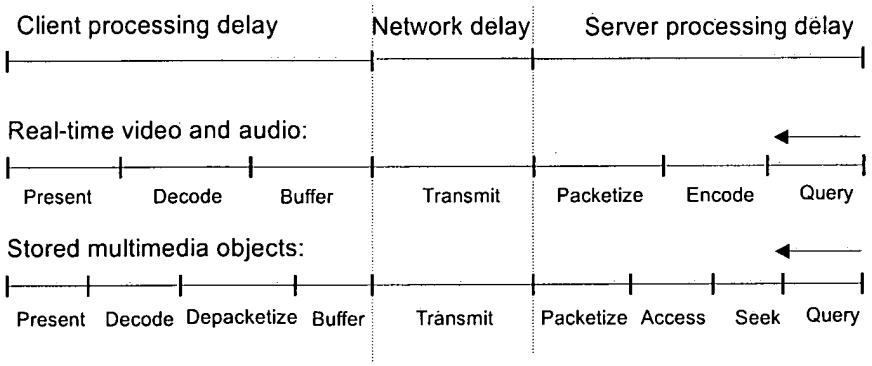

Figure 4: Delay factors from the server to the client, in case of real-time video and audio, and stored multimedia objects [Fuhrt]

It is important that the user gets a fast response to his actions. Long response times can have devastating consequences on the popularity of a service. In [Nielsen] it is argued that a response time of 0.1 second is about the limit for having the user feel that the system is reacting instantaneously. A response time of 1.0 second again is about the limit for the user's flow of thought to stay uninterrupted. The user will notice the delay but normally no special feedback is necessary. The feeling of directly operating on the data is lost, though. 10 seconds is about the limit for keeping the user's attention focused on the dialogue. For longer delays, users will want to perform other tasks while waiting for the computer to finish. Nevertheless, the acceptable response time might vary, since it depends on user expectations and performed task. A user that requests a large image from a remote site, might be willing to wait a bit longer than usually without losing interest. In cases where the response time might be long the user should be given feedback indicating when the computer expects to be done. This is especially important if the response time is likely to be highly variable, since the users will then not know what response time to expect.

4.2.2 Availability

Availability is defined as the percentage of time the system has been available. The availability of a service depends on the reliability of the network components, server(s) providing the service, and the system architecture. The system should be built so that failure of one server or network link cannot cause the service to

become unavailable. Such situations can be avoided by duplicating the service to several servers and having optional network routings to them. In such a system the failure of a server network link only means loss of capacity but the system keeps working.

Since distributed multimedia services offer interactive services whenever the user wants, it should always be available. Or when not, it should recover from faults quickly and with minimal degradation of service. This is particularly important in commercial services.

4.2.3 Reliability

Reliability in a distributed multimedia system means that after a request is submitted, it is accurately responded to without errors.

The server hardware and software must be designed with such carefulness that unexpected events or system overload cannot cause a reduction of the reliability.

4.3 How to get the system to scale better?

To be able to scale a distributed multimedia system its structure and purpose must be thoroughly understood. The system architecture and the characteristics of the provided service and data affect what scaling methods should be used.

In this chapter the nature of multimedia data and different services are first considered. After this, possibilities to scale a distributed multimedia system are presented. The different methods and what kind of data such solutions would suit are analyzed. Further on, critical parameters and implementation ease are considered. Considerations of existing implementations with a deeper analysis are made in Chapter 6.

4.3.1 Characteristics of multimedia data

A multimedia object is usually a composition of several data objects. These objects can be in different data formats, such as text, image, video, audio, etc. The objects are ordered spatially or temporally to create composite multimedia objects. Spatial composition (a, Fig. 5) links various data objects into a single entity, dealing with object size, rotation, and placement within the entity. Temporal composition (b, Fig. 5) creates a multimedia object according to temporal relationship [Fuhrt].

The temporal composition requires synchronization of the data objects. The synchronization can be continuous or point synchronization. Continuous syn-

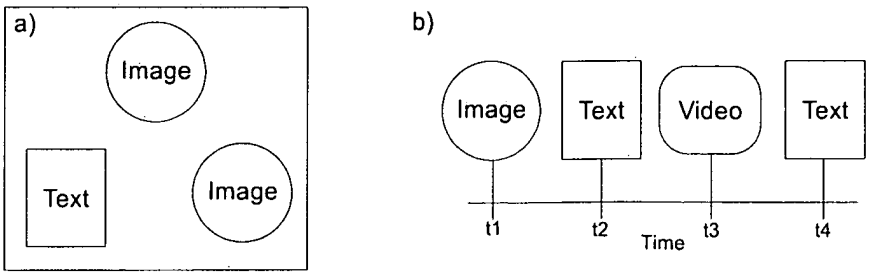

Figure 5: a) Spatially and b) temporally arranged multimedia data objects

chronization requires constant synchronization of lengthy events. An example of this is video conferencing, where audio and video signals are created at a remote site, transmitted over the network, and synchronized continuously at the receiver site when presented. In point synchronization, a single point of a media block coincides with a single point of another media block. An example is a slide show with blocks of audio allotted to each slide.

Two further classes of synchronization are serial synchronization and parallel synchronization. Serial synchronization determines the rate at which events must occur within a single data stream, while parallel synchronization determines the relative schedule of separate synchronization streams.

Multimedia data is often classified according to if it is non-temporal or temporal [Gibbs and Tsichritzis]. Non-temporal data is often called static and temporal data is called dynamic. Static data is for example text, images and graphics. Dynamic data is for example audio, video and animation. With dynamic data is usually also understood data that change according to given input or input time. Such data is for example answers on database queries. To clarify further discussion I will divide dynamic data into two subclasses: non-changing dynamic data and changing dynamic data. With non-changing dynamic data is meant non-real-time audio, video, animation. Changing dynamic data includes real-time audio, video, answers to database queries and other data changing according to given input or input time.

A multimedia object can consist of both static and dynamic data objects. An example of this would be a document describing the weather, with text and a video on the movement of the clouds in the area.

4.3.2 Characteristics of different services

Different multimedia services are based on different system designs and types of multimedia data. This is demonstrated with three examples: video-on-demand, interactive multimedia conferencing and interactive multimedia news.

In *video-on-demand* the customer can select and play movies according to his own choice. The data which is distributed are movies, consisting of a continuous stream of images and sound. These streams must be synchronized internally and intermediately. Since the data is non-real-time it can be stored in advance to strategic servers, according to customer interests. The transportation from the movie repository to the client terminal should be fast and error-free enough to guarantee continuity of the data streams. The customer site requires some kind of buffering for synchronization of the data streams. If the whole movie is downloaded to the customer before viewing large storage capacity is required at the customer site.

In *interactive multimedia conferencing* the conference participants are at different conference sites. The participants have a real-time video connection over which they can communicate. Additionally, they can have a common workspace for sharing information and opinions through text, images, etc. The system can also consist of only the video connection or the shared workspace. Characteristic of the data in such a system is its real-time nature. The data should be delivered between the conference sites without delays. The data should also be synchronized, so that the temporal order is kept correct. Since the data is real-time it cannot be stored in advance to strategic servers. It can only be stored for later reviewing or referencing.

Interactive multimedia news offers the reader news and information. The reader can select among the news material and get more information about items of interest. The system includes databases, where the user can search for old news material. To reduce the information flow the system can make personal versions according to user preferences. The user profile, which defines the structure of the personalized version, could be defined by the user himself or by agents. The agents would follow the user behavior, and shape the service according to what the reader reads. In addition to personalized versions, we have considered the possibility of releasing the media in issues in the OtaOnline project. This would mean releasing time dependent versions.

Such a system would include multimedia data in all possible formats: text, images, video, audio, etc. This would mean that the system includes both static, as well as non-changing and changing dynamic data. The static and the non-changing dynamic data could be distributed in the system according to their popularity. The distribution could be done in times of low network load. Changing dynamic data again requires constant fast and error free transport. The versioning puts requirements on the update mechanism of the system. The whole system should have the same valid version. The personalization adds to the system complexity, since personalized versions must be processed out of the material.

4.3.3 Caching

One obvious solution to get the system to scale better, is to cache data closer to users. Caching is widely used in disk and memory management for holding recently and frequently accessed data. Here, caching means that when a data object is requested from the original site, it is stored at a site closer to the client. When an object is requested, the cache site is first checked. If the object is in the cache it is sent to the client, otherwise the request is forwarded to the original site and the object is sent from there. With proper caching the response time, the network traffic and server load can be reduced.

Such network caching schemes already exist in the Internet and they will be presented further on in this thesis. The network caching has been influenced by experiences gained with hardware and distributed file system caches [Worrell] [Gwertzman].

There are many factors which determine when caching is appropriate. These critical parameters will be reviewed below. After that a closer look is taken at different implementation possibilities and their advantages and disadvantages.

Problems in caching

The caching does not suit all types of data. It makes no sense in changing dynamic data, i.e. real-time video and audio and answers to database queries, for which the contents change according to the time or given input. The non-changing dynamic data can be cached, but requires a lot of storage capacity. The real-time data could, of course, also be stored for later reference, but a cache is not an appropriate place for such storage. In many systems it might be difficult to determine the type of the data, and thus if it can be cached.

Another problem is to decide what to cache. Since the cache size is usually limited it is impossible to cache all objects. As mentioned changing dynamic data should not be cached. To optimize the cache use, the cache should prefer popular objects which stay unchanged for a long period of time. There is no sense in caching objects which will never be requested again or change frequently.

A serious problem of caches is that they might easily serve stale data. The object might change at the original site and the cache site will then have an old copy. There are different methods to try to guarantee that the data is up-to-date.

A very impractical and seldom used method is to define the age of the data object when it is created. That is, the one who creates the object defines how long it will exist without changes. Generally, it might be very difficult to determine the correct lifetime of a data object. In addition, this requires additional work from the creator of the object.

A very simple method is to define heuristic rules how long different kinds of data should be stored before removed or refreshed. That is, the cache administrator could define that files containing GIF images will be stored one day and HTML documents five days. The lifetimes can be based on averages observed at the cache. This method is not perfect, since it can serve stale data. Nevertheless, it is sometimes used since it is simple to implement and requires no checking with the original site of the object.

An approach would be for the object home site to inform all sites when an object has changed. This would require the home site to record to which sites the object has been requested or the message is sent as a broadcast message all over the network. This is quite impractical when there are many cache sites. The management would require a lot of resources and the network would fill up with invalidation messages. There is also no guarantee that the message reaches all sites or that someone requests the data before the invalidation message reaches the site. It is almost impossible to implement an invalidation scheme which would eliminate all stale data.

A widely used method is to ask the original site if the object has changed, when the object is requested from the cache. If it has changed the object is sent to the cache. This approach is quite good. It can cause problems if the original site is not to be reached. In that case stale data can be delivered. It also adds some delays to the object delivery because of the checking, but in exchange the object will be up-to-date.

Complicated invalidation schemes would be required to guarantee total consistency of all caches. The schemes would be so complex and require so much resources, that they would be practically impossible to implement. Thus it is generally accepted that caches can sometimes serve stale data, but this should be avoided as far as possible. Implementations which sometimes can serve stale data are called weakly-consistent.

Caching might also create problems for commercial, confidential or other data objects, which should be distributed to only certain readers. They should not be cached in such a manner that non-authorized persons can access them.

Client caching

One approach is to cache data objects at the client site. This is a good method, if the client tends to access the same objects often. The objects can be accessed faster if it is in client memory and no network traffic is required. A client cache is also easily implemented.

The problem is that this requires the client to have free memory or disk capacity for the caching. Particularly audio and video files can be quite large

and require large cache spaces. Since the probability that the user will access the same object in a short period of time tends to be low, the client caching is not so effective either. It might be a waste of memory resources to cache the objects at every client site. Since all clients still access the object at least once, they will stall a fair amount of network traffic and server load. Client caching can work well for small objects, which are frequently accessed or used in a document. Otherwise, a more appropriate approach is to have a regional cache for some smaller client communities.

Local network caching

In local network caching a cache site is maintained for a local network community. The users define this cache site as their cache. Different from client caching, which works automatically at the client site, a local network cache must be set up and administrated by some central administrator. An additional drawback is that the user must define what cache to use. This can, however, be done with defined system defaults.

When the users request numerous data objects from many different sites the local network cache is certainly more effective than a client cache. The local network cache can offer larger cache space and a higher probability that the object is already cached. This is especially the case, if the users in the community tend to request the same objects. Since the object is more seldom requested from the home site, the network and server load is reduced more effectively.

To implement a local network cache is fairly straight-forward. They are widely deployed in the World Wide Web, for example.

Hierarchical caching

The presented cache solutions create a flat cache hierarchy. When the number of users increase, these solutions will not be able to distribute the load effectively enough. Well working existing network systems, like Domain Name Service (DNS) [Mockapetris87a] [Mockapetris87b] in the Internet, are hierarchically organized. This speaks for implementing hierarchical cache systems. An example of such a system is presented in Figure 6.

Clients belong to some local network community. This local network has a cache, which can be used by all clients. In the same manner the regional network has a cache for the whole region. All local cache sites are aware of their regional cache. The regional caches can again be aware of a national cache. Thus a hierarchical cache topology is created. When the client makes a request, the local cache is first checked. If the requested objects is not found, the regional cache is asked for the object. In this manner the requests follow the cache hierarchy

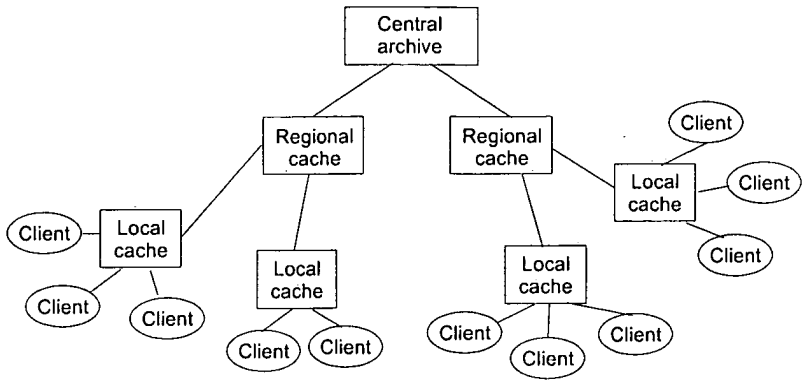

Figure 6: Hierarchical cache topology

until the object is found. If not found in the caches the object is requested from the original site. The cache should also not ask for an object which is, for example, situated in the local network from a regional cache, since this will cause unnecessary network traffic and delay. It should instead request it directly from the home site.

The problem with hierarchical caches is that when the number of traversed caches increases so does the response time. The number of chained caches that will not cause too much delay depends on the system. In [Chankhunthood et al.] it is stated that cache hierarchies as deep as three caches add little noticeable latency for caching in the World Wide Web.

The implementation of a hierarchical cache system requires that the hierarchy is defined and maintained. This requires coordination between cache maintainers in different networks. The caching itself is not complicated to implement. It is the caching hierarchy that requires more considerations. Such hierarchical caching schemes have been implemented, for example, in distributed file systems and the World Wide Web.

A hierarchical cache system based on a fixed cache architecture has a disadvantage. When a cache site is down, the caches beneath cannot access caches above the failed cache.

This problem could be solved with a system where the object is requested from neighboring caches and from a cache a level higher up in the hierarchy as well as the source site and provided from the site responding the fastest. With this method there is a higher probability that the requested object can be provided in spite of a link or node failure. This system also distributes the requests towards cheaper network cost since the closer sites tend to be faster. The disadvantage is the multiple requests for an object, and the network traffic caused by this. Such a scheme has been implemented in the Harvest resource discovery system in the

World Wide Web and will be presented more thoroughly in Chapter 6.

Dynamic hierarchical caching

In a static hierarchical structure, cache and network failures can cause many requests to fail. Thus a dynamic caching hierarchy could be more appropriate. In such a system, the hierarchy could change dynamically according to network and server situation. However, these systems are complex and might be difficult to control. Such dynamic caching hierarchies have been considered in distributed file systems [Blaze].

4.3.4 Replication

Replication means that the data is copied from one site to other locations. It is essentially caching before the data objects have been requested. Replication is a well known way to distribute the load and improve the system performance. For example in the Internet, archive sites are replicated when their popularity increases.

Careful considerations should be made what to replicate and where. To obtain as an optimal solution as possible, the popularity of the data objects should be analyzed. The replicated objects should be placed close to client communities from where many requests are received. When it is not possible to replicate the whole service, only the popular objects should be replicated.

Static and non-changing dynamic data can be replicated easily. But contrary to caching, replication can also be used for changing dynamic data. Instead of replicating only the data, the whole distribution point or service is replicated to another site.

Problems in replication

A critical point of replication is which site the user uses. Usually the user tends to contact the site he knows of. If the chosen site is not the closest one having the requested object, the user might cause unnecessary network traffic. If all users tend to contact the same site, the load is not distributed evenly among the sites. This problem can be experienced at file archives in the Internet. A partial solution would be to define a default site to contact. But the default might not be set, used or it might not point to the optimal site at that specific moment. A scheme where the closest site would be appointed to the users automatically would be an improvement to this scheme. But it is not easy to implement, since it might be cost-intensive to acquire the needed information and to make the decision of the closest site.

With many replicated copies a problem arises how to keep them on the same stand. A copy might change and this change should be distributed to all other copies. The question is how to implement the updating of data objects with as little network and processing overhead as possible. The stricter consistency that is required, the more complicated the system will be.

To be able to replicate, the original site must have permission to copy data to other sites. This requires coordination between system administrators of the different sites.

Simple replication schemes where the data is simply copied in certain time intervals to other predefined locations are quite simple to implement. Most existing systems are based on this approach. Schemes with better consistency guarantees and location of the closest site are under continuous research. Some of the research and proposed approaches will be presented in Chapter 6.

4.3.5 Dynamic request mapping

As mentioned in the previous section it can be difficult to find the best server to contact. Users tend to contact the server they know of, which leads to uneven distribution of load and network traffic.

A scheme where the user request would be mapped dynamically to the best server able to provide the requested object, according to server location, system load, network traffic, etc, would be an improvement. This scheme could be applied to all kinds of data.

It is challenging to implement dynamic request mapping. It has been considered in the Internet, but has not yet been implemented. This scheme will be studied in depth further on in this thesis.

A problem with dynamic request mapping is that it requires all data objects existing in the networks to have a unique id. It is otherwise impossible to locate the requested object from different servers. It might require much work before a common standard on how to implement the id in a distributed network can be reached.

To map the requests according to the current state of the system, system and network load or other system metrics must be available. In some networks, for example the Internet, it might not be easy to obtain this information.

4.3.6 Regulating the quality and price of service

To get the system to scale better for a high number of users, and to decrease the system load, the properties determining the popularity of the service could be

regulated.

The quality of service or the rate of interactivity could be decreased during periods of high load, to accommodate more sessions. Similarly, the fee for using the service could be higher during popular periods of time or for more popular objects [Little and Venkatesh]. This is done in the telephone network for example.

Regulation of quality and price of service is easily implemented and can be done on all kinds of data. The effect is not immediately apparent, though. It is difficult to predict how the users will react to such arrangements. Will it decrease the system load when needed or will it just make the service unpopular and shift the load to another point.

4.3.7 Analyzing access patterns

Prior knowledge about access patterns can help to scale the system. Analyzing access patterns can help predicting the popularity distribution and anticipating the demand of some service. With help of this information it is easier to decide where to place caches and what to replicate where. The information can also be used to determine the off-peak hours, suitable for system management. This includes data redistribution, exchange of management data and system reconfiguration. The analysis can be performed on all types of data.

Analyzing access patterns also helps determining when, where and for how long data must be cached and replicated to minimize storage, usage and transmission costs. Analyzing type and size of requested data can also help to improve the system. The needed network bandwidth and storage space can be anticipated.

This kind of analysis can be easily done by monitoring the system and logging the events. The drawback is that the data can be used only afterwards and it can only predict the coming situation.

The analysis could also be done on-the-fly. The real-time information could be used to dynamically regulate the system according to current needs. For example, when a server suddenly gets overloaded this information could be used to redirect requests. Another example is, if the system detects that many clients in the same neighborhood request the same data almost simultaneously the data could be similcasted to them. To be implemented, the real-time analysis and system regulation require careful planning of the system architecture.

4.4 Comparison of different scaling methods

In Table 4, I have compared the discussed scaling methods. The advantages and disadvantages of all methods are presented. These include critical parameters

and ease of implementation. In case of caching, first some disadvantages common for all different types of caching are presented. After that advantages and disadvantages specific for the different types are presented. The same is done for the method analyzing access patterns.

As stated at the beginning of Section 4.3, the characteristics of the service and the data it uses affect what scaling methods are appropriate to use.

For video-on-demand replication would be appropriate. This because the data exists in advance and could in this manner be effectively distributed. Caching could also be used. In order to know better where to replicate, access patterns could be analyzed. Regulating price and service quality would also suit video-on-demand. Dynamic request mapping is most likely not needed or used, since the customers will be connected to a certain neighborhood server.

Caching and replication would not suit interactive multimedia conferencing, since the data is real-time. Replication could be used so far that data distribution points could be replicated around the network. Analyzing access patterns could be used to help in reserving needed resources in advance or during the conference. Dynamic request mapping has no use since the user will be connected to the closest distribution point.

In interactive multimedia news all presented methods could be used. The methods could be used to complement each other. Caching and replication could be used to bring data closer to the user. Dynamic request mapping could help in finding the most optimal source for each request. Analyzing access patterns could help in decisions of replication, caching and request mapping. Regulating price and quality could also be used.

Method of scaling	Advantages	Disadvantages
Caching		- does not suit real-time data and data changing according to input or input time, e.g. database queries - might serve stale data - can cause problems for commercial, confidential or another data with limited distribution
Client caching	+ simple to implement	- requires cache space - not so effective
Local network caching	+ relatively simple to implement + the possibility of cache hit is bigger	- must be maintained by someone - flat hierarchy limits effectiveness
Hierarchical caching	+ distributes load more effectively, for large number of users	- requires a hierarchy to be defined - deep hierarchy adds to latency - static hierarchy can cause problems
Dynamic hierarchical caching	+ can handle network and server failures better	- complex to implement and control
Replication	+ suits all types of data, since both data and the whole service can be replicated + simple to implement, if not strict consistency between replicas required	- not effective, if not nearest replica used - keeping replicas on same stand, might require much resources - requires permission of local system administrator
Dynamic request mapping	+ suits all types of data + very flexible	- difficult to implement - requires unique document ids
Regulating the quality and price of service	+ suits all types of data + easy to implement	- difficult to predict impact
Analyzing access patterns	+ suits all types of data	
Log file analysis	+ easy to implement	- can be used only for prediction
Real-time analysis	+ results can be directly used	- complicated to implement

Table 4: Summary and comparison of presented scaling methods

5 World Wide Web

There are a few distributed multimedia systems in public use. An example of such a system is the World Wide Web. The World Wide Web[2] (WWW) [Berners-Lee et al. 94] is a large-scale distributed information system, designed to facilitate information retrieval and sharing in the Internet. The WWW design was started at CERN (the European Laboratory for Particle Physics) in 1990 to facilitate the exchange of information between researchers.

One design idea of the WWW is that it offers a single consistent user-interface to numerous different information-retrieval protocols, i.e. FTP, Telnet, NNTP, WAIS, gopher, etc, and their data formats, i.e. ASCII, GIF, Postscript, DVI, TEXinfo, etc. Another idea is that a document can consist of data objects in different data formats including references between documents. The user can move between the documents by selecting such a reference called a hyperlink. To implement these concepts a new protocol, HyperText Transfer Protocol, and a new data format, HyperText Markup Language, was created.

The World Wide Web has rapidly gained popularity. This can be seen in Figure 7. The traffic in the main Internet backbone network, National Science Foundation NET (NSFNET), has been measured by service. The figure presents the percentage of bytes in the NSFNET, for the FTP-data, Telnet, NNTP, SMTP, Gopher and WWW service. As can be seen, the WWW traffic has increased rapidly. At the beginning of 1995 it already accounted for over 25 % of the traffic and due to the form of the curve it can be expected to continue to increase. The U.S government funded NSFNET lost significance as backbone carrier at the beginning of 1995, due to commercialization and will not be one of the major Internet traffic carriers in the future.

The reason for the popularity of the World Wide Web is that it gives a quick and easy access to a large variety of information in remote locations.

5.1 Hypertext and hypermedia

The World Wide Web relies on hypertext and hypermedia as its means of interacting with the users. Hypertext is a non-linear collection of linked items of text. Through selecting a hypertext link, called hyperlink, in a document, the user is taken to another document. In this way the user can move in a free-associative way through information.

If the linked items are of different multimedia formats, such as text, images,

[2]The World Wide Web Initiative: The project <URL:http://www.w3.org/>, Information on WWW <URL:http://www.bsdi.com/server/doc/web-info.html> and World Wide Web FAQ <URL:http://nswt.tuwien.ac.at:8000/htdocs/boutell/>

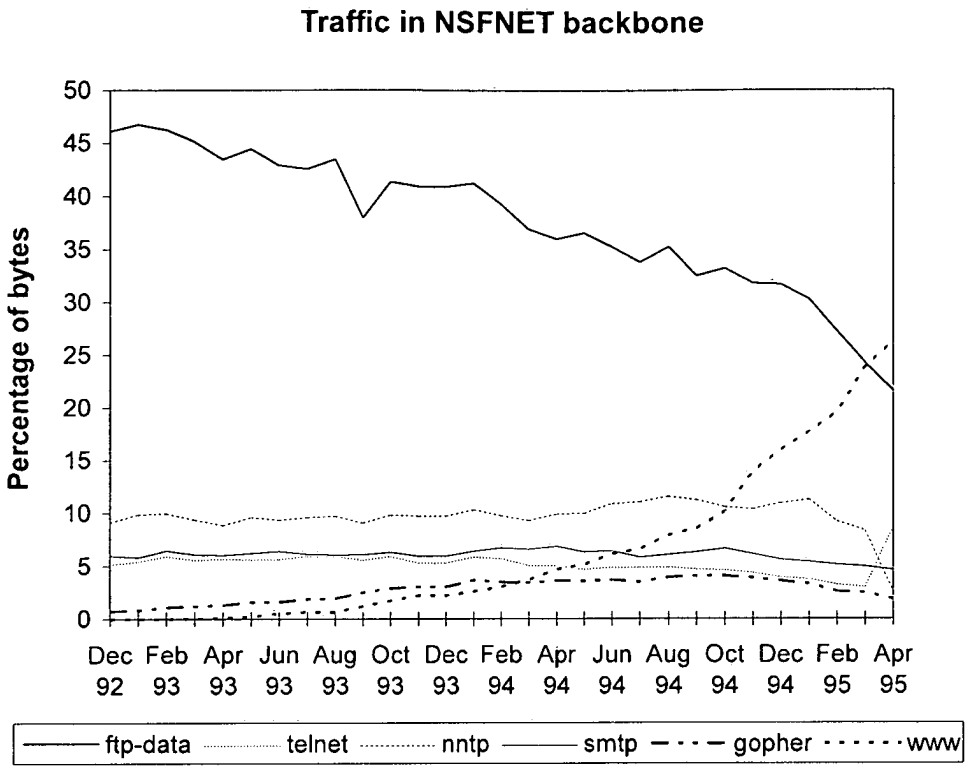

Figure 7: Percentage of FTP-data, Telnet, NNTP, SMTP, Gopher and WWW traffic in NSFNET backbone. Observe that the NSFNET backbone lost significance at the beginning of 1995, due to the commercialization of the backbone

audio or video, the term hypermedia is used.

5.2 HyperText Markup Language

The WWW uses the HyperText Markup Language[3] (HTML) for representing hypermedia documents.

The HTML is an application of the Standard Generalized Markup Language (SGML)[4], a language for describing the structure of documents. A basic idea of SGML and HTML is to ensure that documents encoded according to the definitions should be transportable from one hardware and software environment to another without loss of information.

[3]HyperText Markup Language (HTML): Working and Background Materials <URL:http://www.w3.org/hypertext/WWW/MarkUp/MarkUp.html>

[4]A Gentle Introduction to SGML: <URL:http://etext.virginia.edu/bin/tei-tocs?div=DIV1&id=SG>

HTML documents are standard text files with formatting codes for layout and hyperlinks. Formatting codes for layout include text styles, document titles, paragraphs, and lists.

The current HTML standard, HTML 2.0, supports hypermedia document creation and simple layouting, interactive forms and image maps. A new version, HTML 3.0[5], is still under development. It will support additional features such as, tables, text flow around figures, and mathematical expressions. HTML 3.0 will be backwards compatible with HTML 2.0.

5.3 Uniform Resource Identifiers

To be able to link documents with each other a way to refer and identify documents is needed. For this purpose Uniform Resource Identifiers are used. The Uniform Resource Identifier (URI) is a string which identifies, via name, location, or any other characteristic, a network resource. The URI definition includes two subclasses, Uniform Resource Locators[6] (URLs) and Universal Resource Names [7] (URNs).

The global naming scheme used in the World Wide Web is based on Uniform Resource Locators (URLs). In a URL the identification is based on the location or address of the document. It is possible to reference nearly any file or service on the Internet with an URL.

URLs look like this:

```
ftp://ftp.funet.fi/pub/
http://otaonline.hut.fi:80/otaonline/etusivu.html
news://nntp.hut.fi/alt.hypertext
telnet://alpha.hut.fi
```

The first part of the URL, i.e. the part before the colon, specifies the method of access. The interpretation of the latter part of the URL depends on the defined method of access. Usually the colon is followed by two slashes and then a host name. Sometimes the host name is followed by a colon and a port number, from where to request the service. Further parts may specify the name of the file to request or other parameters to be delivered to the host.

[5]Introduction to HTML 3.0:
<URL:http://www.w3.org/hypertext/WWW/MarkUp/html3/intro.html>
[6]Uniform Resource Locators:
<URL:http://www.w3.org/hypertext/WWW/Addressing/URL/Overview.html>
[7]Uniform Resource Names:
<URL:http://union.ncsa.uiuc.edu:80/HyperNews/get/www/URNs.html>

The URL addressing scheme is based on the location of the resource and is thus not persistent. A naming scheme where the resource could be uniquely identified independent of its current location(s) would provide a persistent naming scheme and transparency of access for network objects. Such a scheme is currently under development and the identifiers are called Uniform Resource Names (URNs).

5.4 How does the World Wide Web work?

The World Wide Web works according to the *client-server model.* To access the WWW the user runs a client program, a browser. The browser interacts with the user and requests documents from a server as the user asks for them. The server again sends the requested document, if the document is found. Most servers also include abilities to run external programs, which can perform small search and other tasks. The server usually sends an answer to the user when the task is completed or at the occurrence of an error.

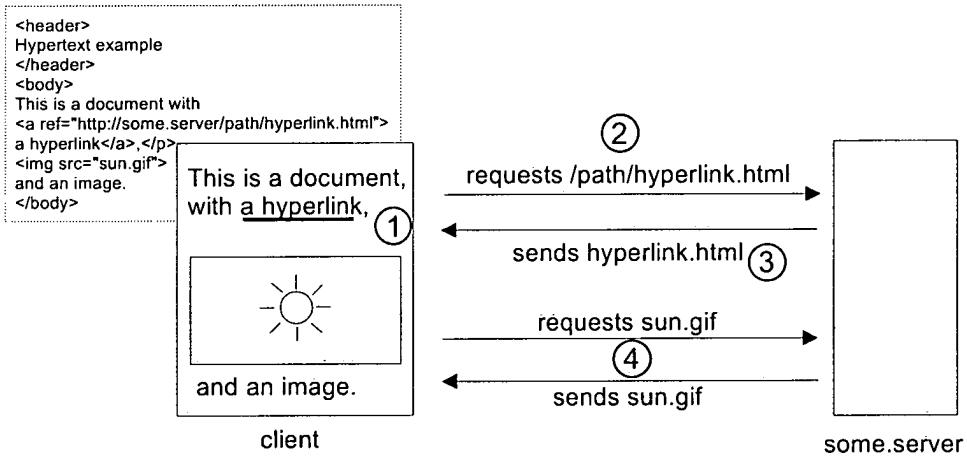

Figure 8: How the World Wide Web works

Figure 8 shows how the World Wide Web works. Running a browser the client chooses a hyperlink in a document (1, Fig. 8). The hyperlink contains a URL. In the figure, the original HTML source text is shown in the left upper corner. The client sees this text only upon request. Using the URL the browser resolves which server to contact and what document to request.

The browser then connects to the specified server through the Internet and requests the given document (2, Fig. 8).

If the document is found the server responds by sending it (3, Fig. 8), otherwise an error message is sent. If the sent document is in HTML, it might include so-called inline images. This means that in the text there are also references, i.e.

URLs, to images. These images will be requested after the browser has received the document. Some browsers open a number of parallel connections, enabling simultaneous retrieval of objects belonging to a document. The above example includes one image. After the browser has received the document it requests the image (4, Fig. 8) and presents it in the correct place in the document.

5.5 The HyperText Transfer Protocol

The language that WWW clients and servers use to communicate with each other is called the *HyperText Transfer Protocol* (HTTP) [Berners-Lee et al. 95]. All WWW clients and servers must be able to speak HTTP in order to send and receive hypermedia documents. For this reason, WWW servers are often called HTTP servers.

The HTTP protocol is based on a *request/response paradigm* and it uses TCP [Postel81b] as the transport protocol.

5.5.1 HTTP transactions

In an HTTP transaction, a client establishes a connection to a server and sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content. MIME (Multipurpose Internet Mail Extension) [Borenstein and Freed] is a mechanism for specifying and describing the format of Internet message bodies. The server responds with a status line, including its protocol version and a success or error code, followed by a MIME-like message containing server information, entity meta information, and possible body content. This is possibly followed by a message body, containing the requested resource.

The HTTP transaction is generated when a user chooses a hyperlink in a document. From the URL in the hyperlink, the browser parses the host id and port number to contact and the URI for the object to request. The default port for HTTP service is 80.

The HTTP includes different request methods. The method "GET" is used for retrieving the resource identified by the URI. With the "HEAD" method only the meta information about the resource is requested. The resource itself is not requested. This method is often used for testing hypertext links for validity, accessibility, and recent modification. There is also a "POST" method, which is used for transferring data from a client to a server, e.g. from a fill-out form in an HTML document to a data-handling process at the server.

A typical HTTP request header generated by the client could look like this:

```
GET /otaonline/etusivu.html HTTP/1.0
Referer: http://otaonline.hut.fi/index.html
User-Agent: Mozilla/1.0N (X11; Linux 1.1.0 i486)
Accept: */*
Accept: image/gif
Accept: image/x-xbitmap
Accept: image/jpeg
Authorization: Basic cRHiaW86LWxlaPJ0eGK=
```

The first field in the request header defines the used request method, the URI of the requested resource, used protocol, and protocol version. In the example the request method is "GET", the requested resource "/otaonline/etusivu.html" and protocol "HTTP/1.0".

The first line in the request header, can be followed by a series of optional header fields. The most common header fields are "Accept", which tells the server which object types the client can handle, and "User-Agent", which gives the implementation name of the client. The field "Referer" allows the client to specify the address of the document from which the URL in the request was obtained. The "Authorization" header is used by the simple authentication scheme used in the HTTP protocol.

The HTTP request header is ended with an empty line. The header is sent to the server and the client then waits for a reply.

On the server side, the request header is parsed, the task defined by the request method is performed and an HTTP reply header is generated. The header and a possible following body, containing the requested resource or request answer is sent to the client and the connection is then closed.

The response header generated for the above request header could look like this:

```
HTTP/1.0 200 OK
Date: Monday, 26-Jun-95 20:00:53 GMT
Server: NCSA/1.3
MIME-version: 1.0
Content-type: text/html
Last-modified: Wednesday, 21-Jun-95 14:31:45 GMT
Content-length: 2719
```

The first line indicates the server's HTTP version and a response code. In this case it is "200 OK", which indicates transaction success. The status line is followed by fields containing information about the transaction, server and the body content.

The "Date" field indicates the time of the transaction. The "Server" field gives the version of the server. The used MIME version is also given in a field. The "Content-type" and "Content-length" are MIME headers which indicate the type and size of the object which follows. The "Last-modified" field indicates when the object was last changed. In most cases it is the timestamp of the file on the disk. For some objects such as executable scripts or search results, no "Last-modified" header is returned.

The client then reads and parses the HTTP reply header, determing the length and type of the data being sent. If the received document contains inline objects, the browser will issue a sequence of requests to retrieve these documents.

Because HTTP was designed to be a lightweight and stateless protocol, it does not currently use a control connection, or keep the data connection open between object transfers.

5.5.2 The performance problems of HTTP

The HTTP protocol has been designed to be an "easy to use" and fast protocol, but there are certain design features of HTTP interacting badly with TCP. This causes problems with performance and with server scalability.

Latency problems are caused by opening a single connection per request, through connection setup and TCP slow-start costs. Further latency is due to the protocol only returning a single object per request [Spero94a].

Let me use an example where an HTML document with one image is requested, in order to explain the problems (see Figure 9).

The client opens a TCP connection to the server, via TCP's *three-way handshake* procedure. The client sends a connection request, the server responds, and the client acknowledges the response.

The client then transmits a request to the server. The server processes the request, and then transmits a response to the client.

This results in a lower bound of two *round-trip times* (RTT) for a transaction. A round-trip time is the time taken to send a packet from one end of the connection to the other and back.

In this example the document includes one image. To receive the image, a new connection is set up and the image is requested. This means an RTT of at least four RTT for the document in the example.

This would be the case if it is assumed that the responses fitted into a single data segment. When TCP transfers a stream of data, it breaks it up into small segments. The size of each segment can vary up to a *maximum segment size* (MSS). The segment size could be negotiated but this is optional. For remote

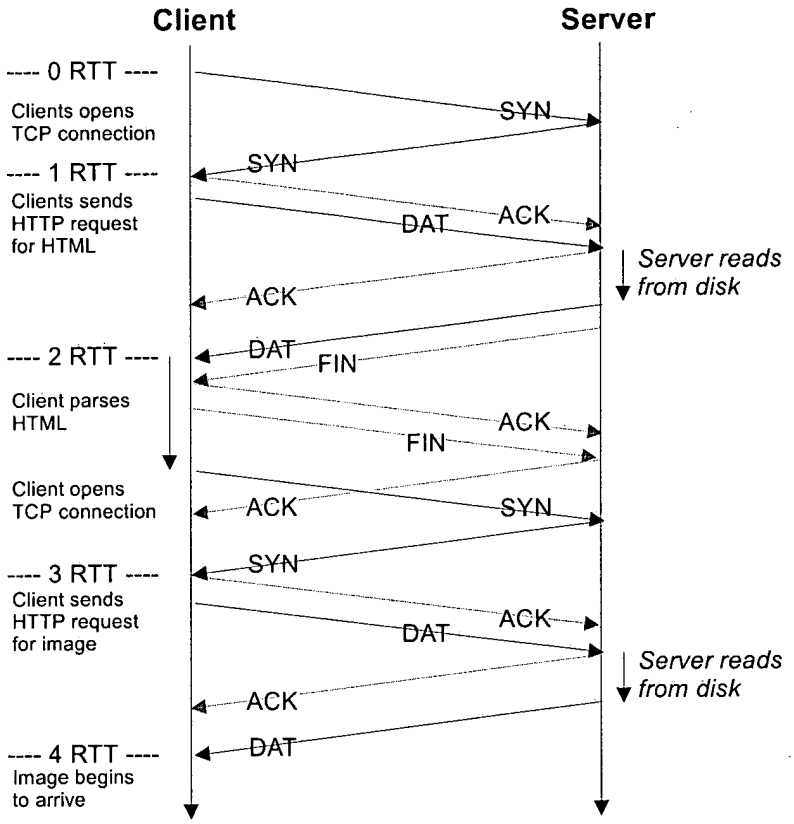

Figure 9: HTTP request example [Padmanabhan94]

connections, the MSS defaults to 536 bytes.

There is an additional latency per connection, because the TCP implementation uses a technique called *slow-start* to avoid network congestion. In slow-start, the sender maintains a window of unacknowledged segments called the *congestion window*. When a connection starts up, each sender is only allowed to have a single unacknowledged segment in transit. When an acknowledgment is received, the congestion window is incremented from one to two, and two segments can be sent. When each of those two segments is acknowledged, the congestion window is increased to four. The window is thus opened gradually, doubling the number of packets each round-trip time. TCP reaches full throughput first when the effective window size is at least the product of the round-trip delay and the available network bandwidth. This approach is good for congestion avoidance in normal connections, but very bad for short-lived connections transferring small documents.

In addition to the latency costs caused by a new TCP connection for each HTTP request, there are other inefficiencies causing scalability problems.

When a server closes a TCP connection, it is required to keep information about the connection for a period of 240 seconds[8]. This means that the server has to leave some resources allocated for every single connection closed in the past four minutes. A busy server could end up with its tables full of connections in this "TIME_WAIT" state, leaving no room for new connections or causing large connection table management costs.

Additionally, the connection setup and teardown requires some processing overhead at both the server and client. This increases the processing load.

The example reflects well how badly the current implementation of the HTTP interacts with TCP. The requested documents are usually of quite small size. Requesting more objects per connection or having longer connections would increase the latency and server load. Considerations on how to improve the HTTP-latency have been made by [Padmanabhan and Mogul] and [Spero94a].

5.5.3 HTTP-NG

To correct the known performance problems in the previous version of HTTP, an improved HTTP protocol HTTP-NG is under design [Spero94b] [Spero95].

In addition to performance improvements it will provide extra support for commercial transactions, including enhanced security and support for on-line payment. These improvements will facilitate the idea of "pay-per-view" hypermedia, a concept which many commercial interests are currently pursuing.

5.6 World Wide Web servers

A WWW server is a program which responds to an incoming TCP connection and provides a service to the caller. The servers are often called HTTP servers because their primary purpose is to provide data using the HTTP protocol. This term is misleading, since the servers mostly also speak other protocols. There are many different types of WWW servers for different platforms available and the number is increasing rapidly. Server software is available, for example, from [WWW] and a comparison chart of different WWW servers is available from [Hoffman].

In addition to providing documents, the server usually includes additional functions. It records the Internet address, time and request made for each connection in a log file. This is done for statistics and security reasons. The server can also protect certain files from non-authenticated users. The server can also execute external programs, Common Gateway Interface (CGI) programs or scripts. CGI is a standard for interfacing external applications with information servers.

[8]This is the recommendation, many implementations violate this specification and use a much shorter time.

CGI applications can be written in a programming language, i.e. C, C++, etc, or in a script language, i.e. Perl, Tcl, etc. Many people prefer to write CGI scripts instead of programs, since they are easier to debug, modify, and maintain than compiled programs. Below I will use the term program for both programs and scripts.

With a CGI program the servers can support additional functions and support datatypes and resources not even conceived of when the servers were invented. For example, with the help of a CGI program requests to a database can be made. Data is gathered by the client, usually using an HTML form, sent to the server along with the name of the program to be run. The server executes the program with the specified data. The program reformats the data, sends it to a database, receives a response and reformats the response as an HTML document. This response is then sent via the server to the client.

5.6.1 Estimating server performance

The server performance has turned out to be a problem at many popular World Wide Web sites. This has led to the development of server techniques which bring better performance. One improvement has been the introduction of servers with multithreading, instead of servers which create a new process for each connection. A thread is lightweight process and consumes less resources than a process. In a multithreading server each connection is handled by a thread.

When the request rate increases the server might crash under the increasing load due to exhaustion of resources in the operating system, typically the number of processes, or the available memory. Even if the server does not crash, it tends to show different anomalous behavior including slow response, dropped network connections and other indications of system degradation. Many of these problems are apparently associated with extremely high rates of opening and closing TCP/IP connections for long periods of time [Katz et al.].

Through analyzing request rates and estimating server performance, performance problems can be predicted and maybe avoided. Performance estimation and testing can also be used for comparing different servers on different hardware platforms for optimal server selection. Such tests can also help to determine the impact on the performance of different security mechanisms, or how efficiently the server can handle different kinds of requests.

The following metrics are usually used to measure server performance:

- requests/sec
- throughput (bytes/sec)
- average response time (per request type)

The number of weekly connections is often used for measuring the average server load, but it does not clearly quantify the maximum performance of the server. A better approximation for the performance is the peak number of incoming requests per second.

Performance estimation and prediction have traditionally been done by analyzing the log files of the server. One problem in this approach is that the log files do not contain all requests. For example, when the load increases and the server begins to deny further connections it has no way to report these failed requests, since it cannot log what it is not receiving.

A better way to estimate server performance is to study how the server reacts to requests sent in a controlled manner. Requests are sent from artificial programs, robots, simulating the behavior of normal clients.

The main problem with such performance test programs is to create a realistic workload on the server. The simplest method would be to randomly choose what and when to request. This totally statistical method is not appropriate, since it does not reflect typical user behavior. A better method is to define a document to start with and then randomly choose one of the hyperlinks in the received document to be requested after a random pause. But since the access patterns might vary from server to server, an optional method would be to analyze log files for user access patterns, so-called user traces, and use these for requesting. The problem with this method is that discharged requests are not logged. Additionally, users are not identified in the log files, if user authentication is not used.

To create different workload representative of different usage patterns, the following workload issues need to be considered:

- request rate distribution
- request type distribution
- file size distribution
- CGI scripts
- protection and encryption

I have implemented a WWW server performance test. It consists of robot programs which make requests to a defined WWW server. The requested objects are chosen randomly among the hyperlinks in the current HTML document or by following a list of URLs in a file. In the former case the robots wait a randomly 0-60 seconds between the requests, in the latter case the time to wait is given in

the file. The robots can be distributed on different computers and maneuvered through a central user interface.

There is also a freely available WWW server benchmark kit, WebSTONE [Trent and Sake]. This standard test was developed to make the comparison of different competitive servers easier. It works in a similar manner to the above mentioned test program.

5.6.2 An example of a performance test

Many different kinds of performance tests have been conducted. Specific for these tests are that the achieved maximum performance varies according to the method used to create the load. Additionally, the results vary according to how the WWW server and the system were configured.

The results of one conducted performance test can be found in [McGrath]. In this test, several WWW servers were tested on an HP 735 workstation, having 96 MB of memory. The test was conducted by programming client programs to repeatedly make requests to the server which was tested. The tested servers were NCSA httpd V1.3, CERN httpd V3.0, NCSA httpd V1.4 and Netsite Communications Server. The NCSA httpd V1.4 was tested in two different modes. In the default mode, where processes are created in advance, i.e pre-forked, and in a mode where processes are created, forked, when needed.

All servers were configured not to use Domain Name Service to reverse resolve client IP addresses. They were also configured to serve documents from the same distributed file system. The NCSA httpd V1.4 in pre-forking mode and the Netsite Communication Server were configured to pre-fork 16 processes. No access control and no CGI scripts or other server side programs were used. The servers only had to respond to HTTP "GET" requests.

The server platform was connected to a 100 Mbit/second FDDI ring. The clients were on an ethernet that connected to the FDDI ring through one router.

The client program was programmed to connect to a target server and port, send an HTTP request, and read the returned data. The request rate was designed to be very intensive. The test program was configured to fetch a 100-byte file repeatedly as many times it could for 7 minutes. One, two, three, four and eight copies of the program were run simultaneously.

One of the main test results is presented in Figure 10. In the figure, the average number of completed connections per second is plotted as a function of the number of running client programs. The test results were received by dividing the number of connections completed during the trial period by the length of the trial.

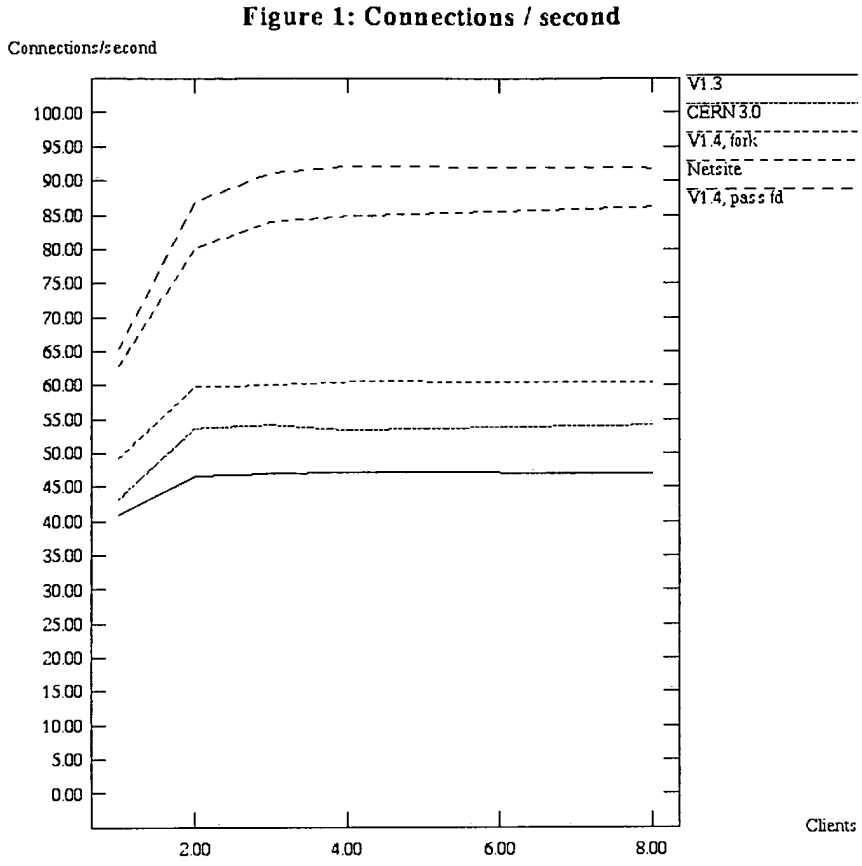

Figure 10: Number of requests per second for different WWW servers running on a HP 735 workstation in the test reported in [McGrath]. The V1.4, pass fd, stands for NCSA httpd V1.4 in pre-forking mode

In this test, the NCSA httpd V1.4, in default pre-forking mode, had a maximum performance of about 90 requests per second and the Netsite Communication Server had a performance of about 85 requests per second. The other servers had a considerable smaller performance.

These test results should only be seen as an example, since the results might vary according to how the test is conducted and how the system and server have been configured.

5.6.3 The performance of the OtaOnline WWW server

In this chapter the performance of the OtaOnline WWW server will be analyzed. The current performance needs will be compared to the maximum available performance. To better understand the current performance needs, the characteristics of the OtaOnline service will first be presented.

The OtaOnline service is provided by using the NCSA httpd V1.4 server software. The WWW server is running on a DEC Alpha 1000 server, having 128 MB memory. The server is connected to the local network of the Helsinki University of Technology. The server software is configured to pre-fork 10 processes and to serve the material from local disk.

The access to the OtaOnline service is restricted to the campus area of the Helsinki University of Technology. OtaOnline requires users to register before they can use the service. Since the start of the service, the number of registered users has been steadily increasing (see Figure 11).

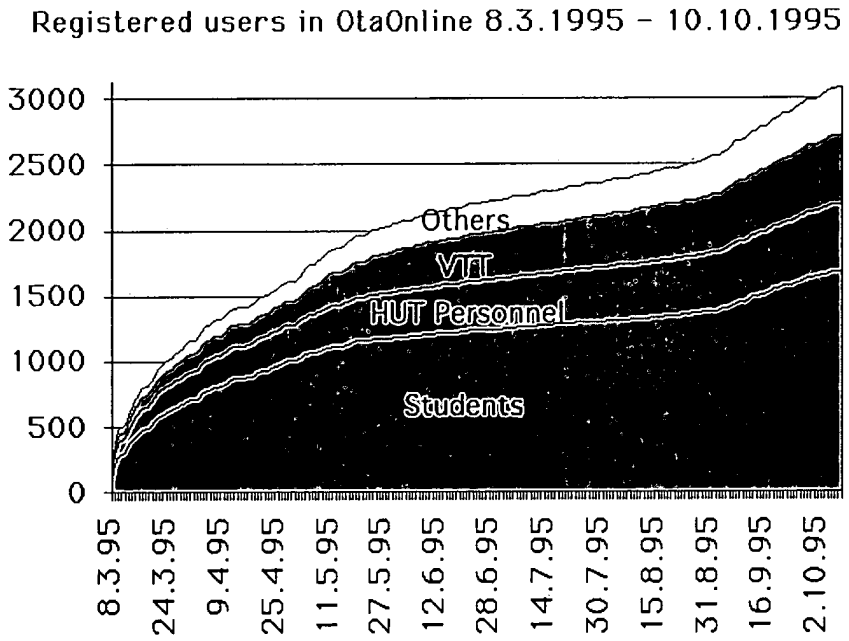

Figure 11: Number of registered users in OtaOnline, specified according to different user types

The material distributed in OtaOnline is received directly from member newspapers within the Aamulehti Yhtymä, Ltd and the Finnish Meteorological Institute. Some material is produced by local editors. At the moment, the material includes HTML documents, and images. The update frequency varies from three

hours to weeks depending on the type of the material. Audio and video material has been used only sporadically, since we have been concerned about the performance bottlenecks these data types might cause. Currently, OtaOnline includes no audio or video material, but we intend to use these to a higher extent in the future.

Some figures characterizing the OtaOnline service is presented in Table 5. The figures were observed in October 1995 and they are not constant.

OtaOnline characteristics	
Total number of files	470
HTML documents	200
Image files	227
Other files, including CGI scripts	63
Average size of HTML documents in bytes	5533
Average size of image files in bytes	18424
Average number of images in an HTML document	3.5
Registered readers	3244
Reading sessions/week	2230
Different readers/week	929
Average number of times a reader reads per week	2.4
Average number of HTML documents requested per reading session	19
Average number of all files requested per reading session	85

Table 5: Characteristics of the OtaOnline service

The load of the OtaOnline server varies depending on the day of the week and the time of the day. On the whole, the load of the server is increasing, since more and more users use OtaOnline. During the most busy hour of the day up to 50 requests per 10 seconds have been recorded (see Figure 12).

To determine what kind of load the current OtaOnline server can sustain a performance test was conducted. The test was performed with help of the robot programs I had written. Robot programs were started on 13 different workstations in the local network of the Helsinki University of Technology. Eight of the workstations were situated in the hut.fi domain and five in the cs.hut.fi domain. On each workstation, 10 robots were successively started. The robots simulated users which first requested the front page of the OtaOnline service and then randomly followed a hyperlink in the received document. The robots paused randomly 0-60 seconds between the requests. The created load was higher than normal OtaOnline users create, since the pause between requests normally varies between 0-180 seconds. During the test system information of the server was gathered with the monitor program.

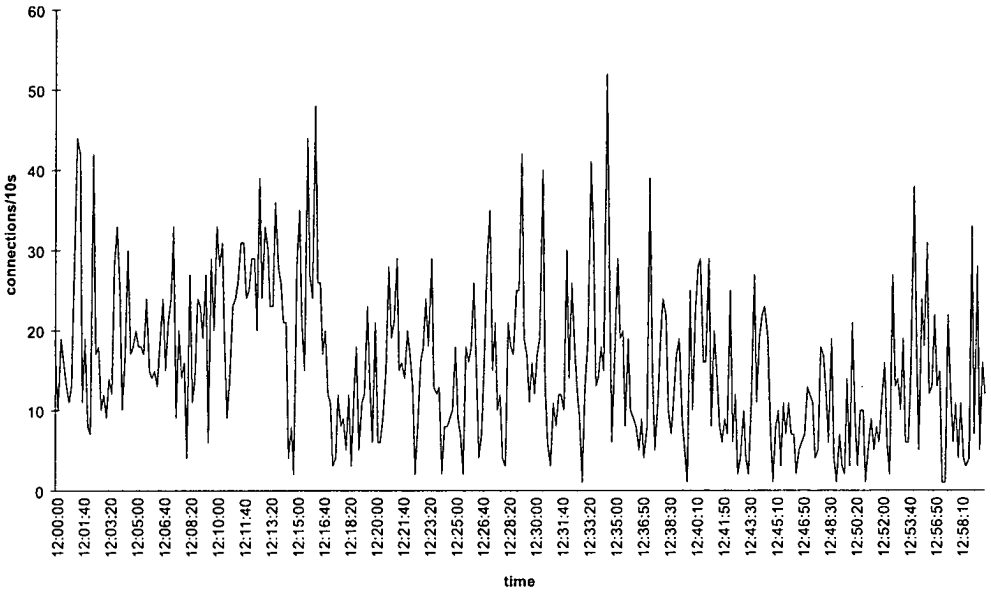

Figure 12: Connections per 10 seconds on the OtaOnline server during the busiest hour of the day

The results of the performance test are presented in Figure 13 and Figure 14. After the test was started, the number of requests to the server increased gradually, as more and more robots were started (Fig. 13). The variation in the number of requests or served connections is caused by the random request rate of the robots. After some time, the request rate stopped to increase, even though additional robots were started. Instead, timeouts started to occur. A timeout indicates that a request was not received by the server due to the server or network resource exhaustion. As can be seen Figure 14, at the same time as the number of connections stopped to increase, the idle time of the server CPU decreased to 0. The CPU measurement was started when the test had been running for a while. The decrease in requests at the end of the test is caused by the fact that robots were stopped.

The results of the test indicate that with the current configuration the Otaonline server can serve up to 20 requests per second. Compared to the current performance needs of the OtaOnline service, the number of requests per second can quadruple without performance problems.

The OtaOnline server performance of 20 requests per second is about a fourth of the performance reported in [McGrath] for the NCSA httpd V1.4 server. The results cannot be directly compared though, since the WWW server was run on different systems and the tests were differently conducted. One important difference is that in the test reported in [McGrath], the same 100 byte file was repeat-

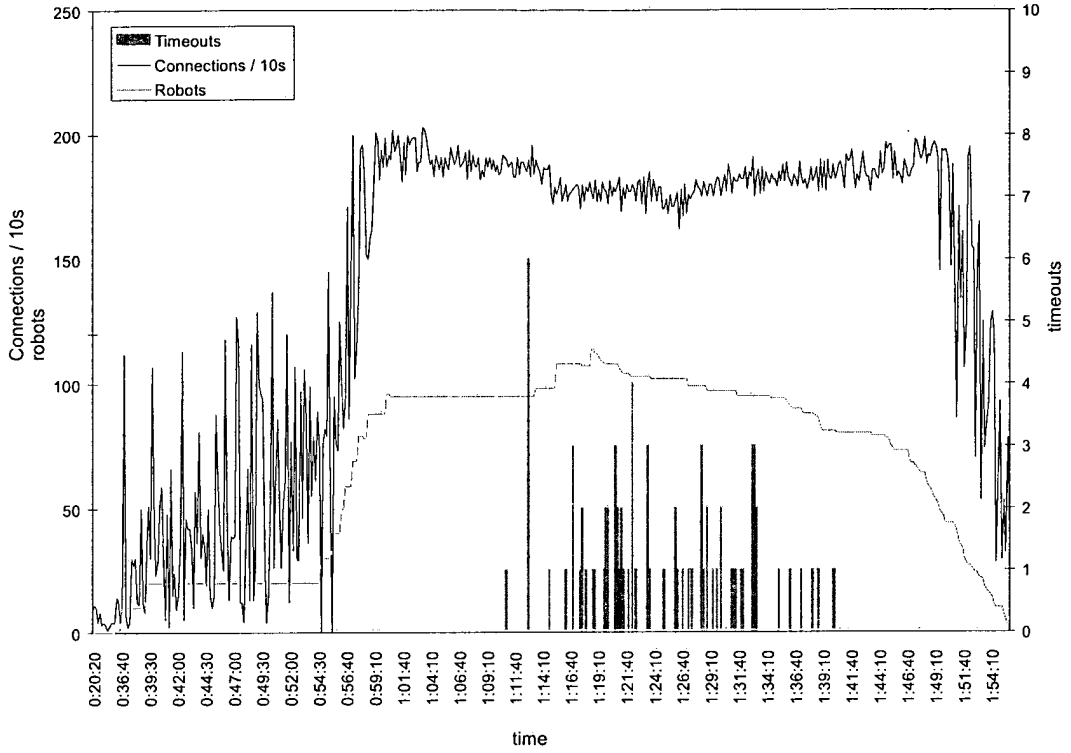

Figure 13: The number of connections per 10 seconds, the number of robots, and the number of timeouts in the OtaOnline performance test

edly requested. In the OtaOnline test, the requests were randomly distributed among all available files. The average size of all requested files was 3 kB. Thus more time was spent on looking for and sending the files, as in [McGrath]. Contrary to [McGrath], OtaOnline also uses user authentication and CGI scripts, which both require processing capacity. Additionally, less pre-forked processes were used in the OtaOnline server.

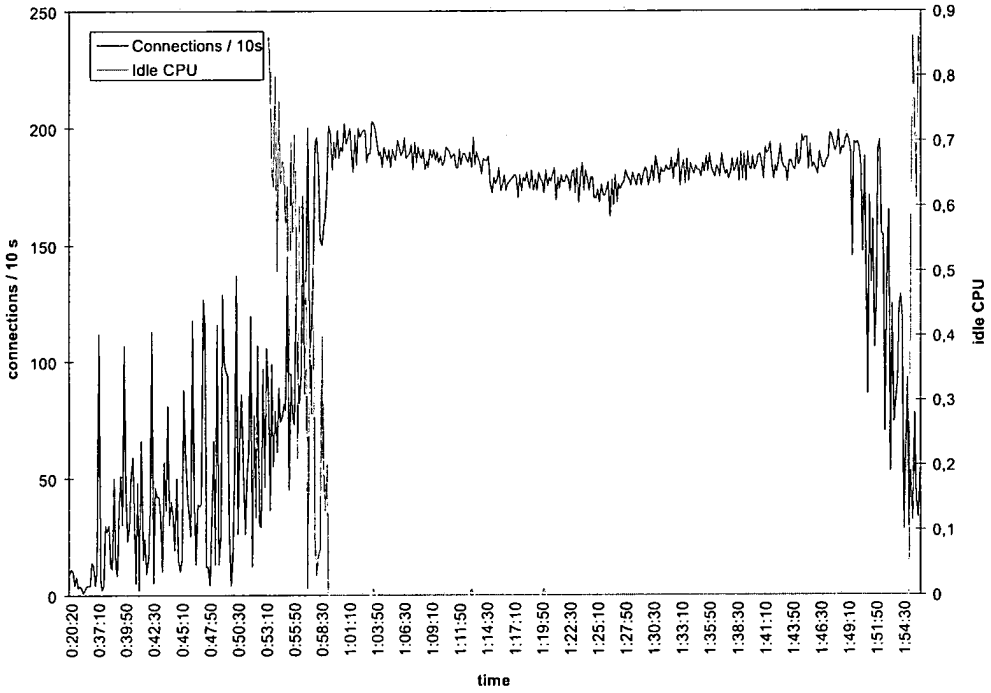

Figure 14: The number of connections per 10 seconds, and the idle time of the server CPU in the OtaOnline performance test

6 Scalability in the World Wide Web

The growing number of users and the increasing size of the data objects are the most significant factors which cause scalability problems for the World Wide Web. The popularity of the World Wide Web has grown almost exponentially in the last two years and the growth can be expected to continue (see Fig. 7). The use of bigger and more complex data objects, i.e. audio and video, has also increased. These two factors lead to higher network traffic and server performance bottlenecks. Unnecessary network traffic is also caused by the inefficiency of the HTTP protocol. When using the World Wide Web, long response times due to network and server performance can be experienced almost daily.

To improve the situation better performing servers and more network capacity could be added. The network bandwidth could be used more effectively by improving the HTTP protocol and by using data compression methods. But since free capacity tends to fill up and these methods do not distribute the load, additional methods have to be considered to create a scalable system.

The most popular methods to scale a World Wide Web service have been different caching and replication schemes.

This chapter tries to give an overview of current scaling methods in the World Wide Web, along with related research and planned implementations.

6.1 Cache systems

In the World Wide Web caching is the most used method for achieving better performance. Caching schemes for the Internet and WWW have been widely considered and tested.

Danzig et al. have studied the amount of traffic caused by FTP in the Internet. They show through simulation that the FTP-traffic could be reduced by 42% if caches were installed at strategic spots in the Internet [Danzig et al. 93]. The high percentage can be explained by the fact that a small set of the transferred files were very popular and often requested. Danzig et al. state that the same concept could be used for other internetwork services, such as WWW, with similar results.

Bestavros et al. compared caching at the session level, the host level and the local area network (LAN) level [Bestavros et al.]. In session level caching, caches for separate sessions are managed independently. In host level caching, caches for separate hosts are managed independently. In LAN level caching, caches for separate LANs are managed independently. Their results show that while session level caching can be nearly as effective as host and LAN level caching, it consumes much more resources. For a given level of performance, less system

resources are consumed by host level caching, and even less are consumed by LAN level caching. Thus if a fixed amount of resources is to be allocated to caching, they are best allocated to LAN level caching. The results affirm that caching is most appropriate on local area network level.

WWW traffic studies conducted at different WWW sites, among others [Bestavros et al.], [Braun and Claffy], [Glassman] and [Sedayao], show that caching is a very suitable scaling method. This is justified by the fact that at many sites a small number of documents stand for most of the requests. Additionally, there is a strong preference for requesting small documents. At least in [Braun and Claffy] a wide geographic diversity of request sources was also experienced. These three factors speak for deployment of geographically distributed caching mechanisms to improve server and network efficiency. Naturally, there are differences in file sizes and request distributions at different sites. These highly depend on the offered service and the users, but the mentioned tendencies are widely observed in the Internet.

The National Center for Supercomputing Applications' WWW server received 837,046 requests in a period of two days [Braun and Claffy]. The 25 most popular documents were responsible for 59% of the requests and 45% of the bytes sent. The average size of a requested file was 18 kB. The site had 8,494 unique documents. Of the documents 95% were smaller than 60 kB.

In [Glassman] the requests to multiple sites from users residing in one network domain were analyzed. The average size of a requested file was 14 kB and the size of the unique files requested were distributed according to Table 6.

Size (kB)	Percentage of files
-1KB	25%
1-5 KB	41%
5-10 KB	13%
10-50 KB	14%

Table 6: The size of files requested through the WWW at the Digital Equipment Corporation [Glassman]

The strong preference for small files is surprising due to the potential for multimedia content in WWW documents and the large amounts of data needed to transfer images, video, and sound. However, a number of factors may tend to increase the proportion of small files. Firstly, many of the images included in HTML documents are actually fairly small in size. Small images, such as buttons and icons, are often used in the documents. Secondly, users may tend to interrupt document transfers which take too long [Bestavros et al.].

Even though the OtaOnline service can be accessed only locally, it shows similar tendencies as the above mentioned sites. When analyzing the access log

file from the 4th of September the 25 most requested documents were responsible for about 50% of the requests and 26% of the bytes sent. The average size of a requested file was 8 kB. Of the documents 92% were less than 50 kB. Table 7 shows the file size distribution.

Size (kB)	Number of files	Percentage of files
-1	26	11%
1-5	99	43%
5-10	41	18%
10-50	46	20%
50-100	15	7%
100-	3	1%

Table 7: File size distribution in the OtaOnline service on the 4th of September 1995

6.1.1 Client caching

Client caches are widely used in the World Wide Web. Almost all current WWW browsers are provided with a cache. This includes disk as well as a memory cache. The cache size can usually be defined in the browser, but typical default values are 5 MB and 2 MB, respectively. The caches work as session caches, with the difference that the objects cached on disk, will be saved from one session to another.

The problem with client caches is that they easily serve stale data. If the requested object is found in the cache, it is often served without checking if it is still valid. Some browsers are designed to check with the original site if a document has changed, but this does not seem to be working properly. Most browsers provide a reload command, to enable the browser to receive the latest version of a document. When reload is selected the shown document is requested from the original site. The replacement algorithm used when the cache is full, varies in different browser implementation. An algorithm where the least recently used objects are removed seems the most appropriate approach.

Client caches are efficient only if requested objects are small and frequently accessed. For users with slow network connection client caching is recommended, since it brings increased performance. However, since the cache reserves local disk space and might serve stale data, many users tend to turn the caching off, especially if a local network cache can be used instead.

6.1.2 Local network caching

As shown in [Bestavros et al.] caching in a local area network is more effective than client caching. Disk space is saved, since only a single copy needs to be cached. In a local network cache the probability that the object has already been cached is, additionally, higher. In the WWW local network caches are widely used. They are implemented in so-called proxy servers.

Proxy servers

Proxy servers [Luotonen], or proxies for short, were first designed to enable a standard method to access the Internet from behind a firewall. A firewall is a machine providing access to the Internet from a closed subnet. Since the same proxy is usually used by all the clients within a subnet, it was realized that efficient caching could be implemented in it. The first caching proxy was the CERN httpd[9] and it is the most widely used. Other caching proxies are Lagoon[10] and Harvest[11].

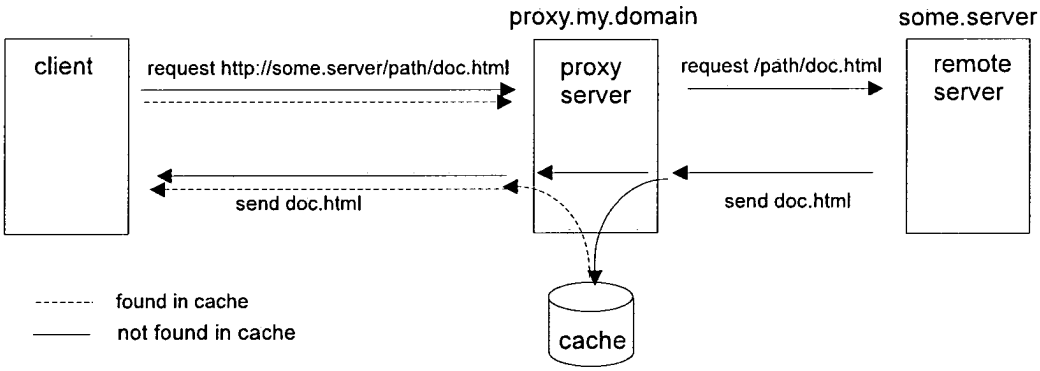

Figure 15: How a caching proxy server works

Figure 15 describes how the caching proxy, hereafter simply referred to as proxy, works. The default proxy is defined by the user in the browser or in an environment variable. When the proxy receives a request from a client, it first checks if the specified object already exists in the cache. If not, the object is requested from the remote server specified in the URL. In the request the proxy uses the header field it received from the client without modification. When the object has been retrieved from the remote server, it is cached on disk and sent to the client. If the requested object is already in the cache, the server must

[9]CERN httpd <URL:http://www.w3.org/hypertext/WWW/Daemon/>
[10]Lagoon <URL:http://www.win.tue.nl/lagoon/>
[11]Harvest <URL:http://harvest.cs.colorado.edu/>

determine if it is up-to-date. If the object is up-to-date it is sent, otherwise it is first requested from the remote server and then sent to the client.

In trying to always offer the latest version of an object the CERN proxy sets an expiration time when the objects is cached. This the time when the object will be refreshed or deleted from the cache. The CERN proxy first looks for an "Expire" line in the HTTP response header. "Expire" is seldom used, though. After that, it looks for a "Last-Modified" line in the header and calculates an expiration time from the time the object was last modified. Default expiration times can also be specified in the server configuration file.

If the object requested by the client has expired but is still in the cache, an "If-Modified-Since" request is made to the remote server. If the server answers with a "Not-Modified" message, the cached object is sent to the client. If the object has been modified, it is fetched from the remote server and then saved on disk and sent to the client.

The CERN proxy can be configured to guarantee that the cached objects, which have not expired, are up-to-date within some amount of time. In this case, an "If-Modified-Since" request is sent if the time since the requested object was last accessed in the cache exceeds a defined threshold.

It is a configuration option of the CERN proxy whether or not to remove expired objects from the cache. It can be configured to remove objects when they have reached a certain age or when they have been unused for a certain time. Objects are removed when the cache size limit is reached, or periodically to keep the cache smaller.

Other proxies are working similarly to the CERN proxy. The "If-Modified-Since" scheme is a good solution to check if the document is up-to-date, but it causes some extra network traffic. Sometimes it might also be impossible to reach the original site. In this case, the cache offers the version it has.

Proxies may cause problems for objects whose distribution is restricted. This is caused by the fact that all requests coming via a proxy will be identified as comming from the proxy. If someone outside can contact a proxy inside the allowed domains, he can also request the access restricted objects. Proxies will also cause problems when analyzing request distributions. The request will be logged coming from the proxy and not from the host from where the request was originally made. If an object is served from a cache, the WWW server administrator at the original site will not have an idea of the total interest in the object, since the request is not registered at the original site.

The efficiency of proxy caches has been analyzed. Typical reported hit rates for a cache are 30-60% of the requests, depending on the cache size and request distribution [Smith] [Pitkow and Recker] [Glassman]. As Pitkow well observes, the existing caching schemes are still mostly based on arbitrarily defined hard-

coded parameters. These parameters include, for example, when to perform garbage collection, the life time for cached files, and the maximum file size allowed in the cache. Pitkow proposes that an ideal caching algorithm should flexibly adapt its parameters. Thus, as hit rates and access patterns change, the number of documents, the size of the cache, and the actual documents in the cache should also change. This suggests that an empirical analysis of document access patterns may provide the basis for developing adaptive caching strategies.

Web-proxy

To solve some of the problems occurring in proxy caching, Wessels has designed an enhanced proxy, a *web-proxy*, where most of the cache management functions are implemented separately from the proxy [Wessels]. The improvement compared to traditional proxies is that documents could be assumed to be reasonably current without sending an "If-Modified-Since" request. The cached objects would be registered at the remote site they origin from. This site could then notify the web-proxy when an object should be updated. The web-proxy would also ask for permission from the remote site before caching an object. The system was also designed to enable feedback statistics on cache hits to be provided to the objects' original sites.

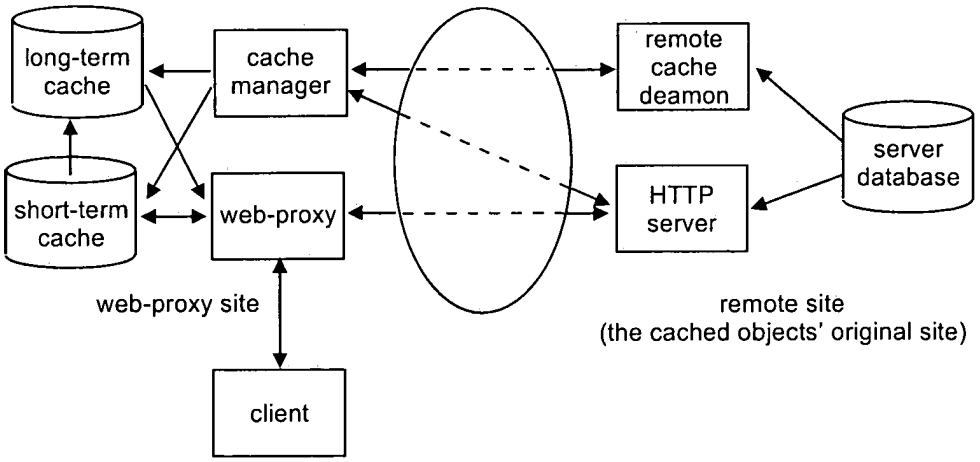

Figure 16: Architecture of the web-proxy

The architecture of the web-proxy scheme is presented in Figure 16. The web-proxy uses two distinct caches. It copies all eligible objects into a short-term cache. Objects with no "Last-Modified" header are not cached, because they tend to be from scripts. Objects that require user authentication or are not old enough are not cached, either.

The cache manager maintains the objects in the short-term cache. It will

remove the objects from the short-term cache when they have reached a certain age. The threshold age can vary depending on the type of object. The threshold is determined from a table, which can be configured. A clean-up program, removing old objects, is run at certain time intervals defined by the administrator.

Some objects get selected for the long-term cache by the cache manager. The cache manager analyzes the web-proxy log files at regular intervals and makes a priority list of candidate objects to be in the long-term cache. Before an object is put into the long-term cache, the manager contacts the cache daemon running at the object's original site and asks permission to cache the object. The administrators at the original site have configured which objects are allowed to be cached. If permission is granted, the object is put into the long-term cache. If it is no longer to be found in the short-term cache, it is retrieved from the original site. After this, the object is registered with the remote cache daemon. The daemon periodically scans the registered objects on the local disk. When an object has been modified or deleted, the remote cache daemon notifies the web-proxy's cache manager. The cache manager can then either remove the cached object or retrieve an updated version from the original site.

A serious problem with this application is that it requires the installation of the cache daemon program at remote sites. If run without the daemon, a remote site cannot explicitly grant or deny permissions to cache objects. The remote site cannot either initiate callbacks to the cache manager when objects are updated. It will probably prove quite difficult to get a significant number of sites to run cache daemons.

6.1.3 Hierarchical caching

Hierarchical caching has widely been considered as an enhancement to the previous described caching schemes. Arranging caches hierarchically according to network topology, might help to distribute the load and reduce access latency.

It has been suggested that proxies could be chained in a hierarchical topology. In case a proxy does not have the requested object, it would request it from another proxy instead of the original site. When the number of chained proxys increases, the cache in top of the hierarchy becomes responsible for more and more clients. If many requests fail on the lower levels of the cache hierarchy, the root cache can be overloaded with requests. The response times can also grow unacceptably if the hierarchy is very deep.

It has also been suggested that different caches would fulfil different purposes. The proxy would on a cache miss, make the request to the cache most likely to give the best response. It might, however, be problematic to know which cache is the most appropriate.

There is one working hierarchical caching approach in the WWW, the Harvest cache subsystem.

Harvest cache subsystem

The Harvest [12] [Bowman et al. 94a] [Bowman et al. 94b] is a system that provides an integrated set of tools for gathering, extracting, organizing, searching, caching, and replicating information across the Internet. The system interoperates with WWW clients and with HTTP, FTP, Gopher and NetNews information resources. Harvest includes a hierarchical caching subsystem.

The caching subsystem is built up so that each hierarchy level contains several neighbor caches. All neighbor caches on the next higher level in the hierarchy are called parents of the caches on one level below. The idea is presented in Figure 17. In this example the neighbor caches are on the same network. The caches in the backbone network are parents for the two groups of neighbor caches at the local area network level.

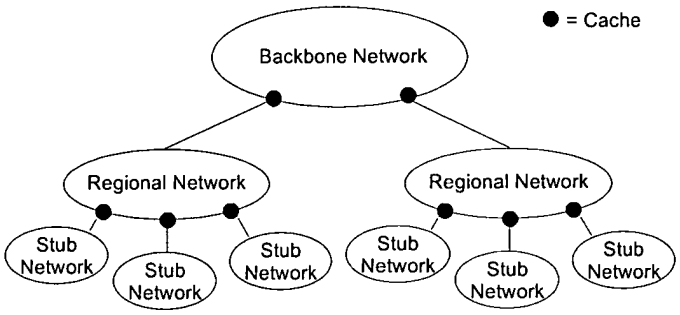

Figure 17: Hierarchical cache topology

The client accesses a cache in the cache subsystem in the same manner as a proxy-server, i.e. by defining the proxy-server to use in the browser on in a environment variable. If a query to a cache fails, the cache queries all its neighbor caches, its parent caches and the object's home site. Each neighbor and parent cache responds with a hit or miss message, and if the object's home is running a UDP echo daemon, it echoes a hit message as well. The cache then fetches the object from the fastest site to return a hit message, whether it is another cache or the object's home site. If all caches send a miss message and the home site is slower than all parent caches, the cache retrieves the object through the fastest parent cache to miss. Otherwise, if the home site is almost as fast as the fastest cache, the object will be requested from it.

A disadvantage of the Harvest cache subsystem is that the cache hierarchy is configured manually by using configuration files. A Harvest cache can also be

[12] http://harvest.cs.colorado.edu/

configured to run alone. In that case it works in a manner similar to a proxy-server.

6.2 Replication

Replication is a popular method for distributing the load in the Internet. An example of this is the network news NNTP [Kantor and Lapsley] and the mirroring of FTP [Postel and Reynolds] archives.

In the World Wide Web replication has been used quite sporadically. But since the network and server load has increased rapidly, many sites have started to mirror their services to areas from where a lot of their requests come. Mirroring means here that a WWW server is set up at another location and the data from the original site are copied to the new server. Whenever a document or data object is changed, it must also be updated at the mirror sites.

It is challenging to develop replication schemes that scale well, use the network topology effectively when updating replicas and are able to keep the replicas consistent. Another challenge in replication schemes is to locate the nearest server having the wanted object. If the server is chosen randomly, network traffic is certainly not optimized.

6.2.1 Demand-based replication

Bestavros has considered disseminating the most popular documents to different servers [Bestavros 95a]. In his proposal documents would be automatically disseminated according to their popularity. In order to do so, each server must collect statistics on the popularity of each document it maintains.

To make the dissemination more effective, servers would maintain a list, mapping local URLs to URLs of corresponding disseminated copies. If the requested document has been disseminated, the server could return the URL of the most appropriate situated copy to the client. The client could cache such redirects to other copies and use them later on, instead of the original URL.

Since at many WWW sites remote access is confined to a very small subset of documents, demand-based replication could be a good approach. The server load can be reduced, while network load is balanced.

6.2.2 Geographical push-caching

A similar approach to demand-based replication is the technique termed *geographical push-caching* proposed by Gwertzman [Gwertzman]. In geographical

push-caching the servers use their knowledge of geographical proximity and access patterns to distribute their popular objects among other geographically distributed servers. Geographical information is used instead of topological information, since information about network topology is not widely available in the Internet.

The system consists of modified HTTP servers, hereafter called push-cache servers, and a registry. In addition to serving clients, a push-cache server is responsible for tracking geographical access information for the objects it provides and for accepting and offering replicas obtained from other servers. The registry keeps track of push-cache servers that are willing to serve replicated objects.

When the demand for an object exceeds a set threshold, the push-cache server will try to replicate it to another server. To do this the server contacts the registry. The registry returns a list of available servers. Based on the list and the collected access patterns, the server then makes the decision to which server to replicate.

The client would contact a push-cache server as a normal HTTP server. The server may then redirect the client to a closer server. When a replica is requested from a server, its original site will be contacted for checking its consistency. If the object has changed, it will be requested from the original site. These two factors might cause a bottleneck at the original server of the replica.

Regarding scalability, the registry is a critical point. It is essentially a database of every available push-cache server, and it must be able to handle requests arriving constantly from all push-cache servers. Gwertzman states that the need for scalability and reliability implies that the registry must be a widely-replicated distributed database. He states that since it is meant that the registry selects lists of servers at random, a given instance of the database does not have to know about all the available servers in order to provide a representative sample.

Gwertzmans simulations led to the conclusions that push-caching is most effective for popular servers which have a small set of popular documents. He also concludes that push-caching is more effective in decreasing the load of the primary host than proxy caching. But on the other hand, proxy-caching saves significantly more network bandwidth.

On the whole there are many open questions in the push-caching scheme that still have to be considered.

6.2.3 Flood-d

Danzig et al. state that current replication schemes used in the Internet are not scaling well, because they manage single, flat groups of replicas [Danzig et al. 94]. They might work well for applications with up to 30 replicas which reside within a local network, but they are insufficient for wide-area, massively replicated services

whose replicas are spread throughout the Internet. Furthermore the replication topology is usually defined by the system administrators, causing them to be static and vulnerable for node and network link failures.

To improve the replication, Danzig et al. have designed a scalable replication scheme, called *flood-d*. It is designed to scale for thousands of weakly-consistent replicas, i.e. for replicas which might not always be consistent. Flood-d is used in the Harvest system for managing replicated data.

In flood-d, sites with replicas are organized into hierarchical, autonomously administrated replication groups (see Figure 18). The flood-d floods updates according to the logical update topology between the group members. For each replication group, flood-d calculates an update topology that attempts to minimize the network cost and propagation time needed to transmit updates. When a site wants to inform about an update it floods the data to its logical neighbors in the update topology. To forward the information to other replication groups, there are so called corner members which are connected to another replication groups. When an update is sent the flood-daemons running on each site measures the available bandwidth and round-trip time. This information is used when calculating the update topology.

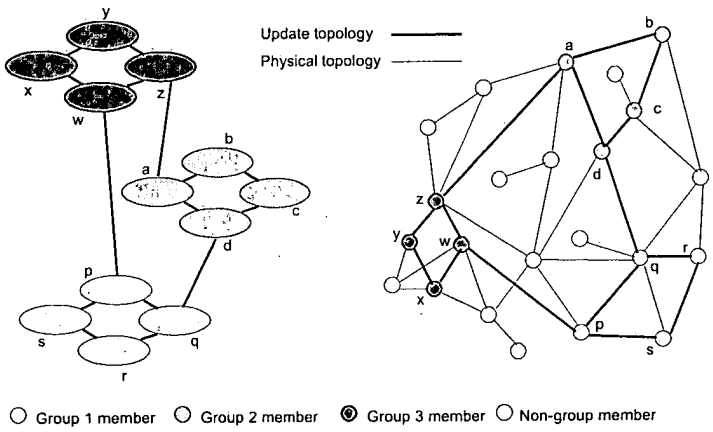

Figure 18: Replication groups in flood-d, showing logical versus physical topologies

A hierarchical organization of the replicas limits the amount of consistency information each replica needs to keep. Each replica only needs to make sure that all members of its replication group receive an update. The corner replicas are an exception since they also have to ensure that an update is received by the members in the neighboring replication group. This arrangement also minimizes the time to reach a consistent state.

When a new member joins a replication group or in periodic intervals the update topology is re-calculated and flooded to all group members. According

to Danzig et al. any flood-d replica can initiate the topology calculation. The approach in Harvest uses a group master, which periodically computes the logical topology [Bowman et al. 94b]. This leader cannot be re-elected. If the leader fails, replication proceeds but the logical topology is not updated and new replicas cannot join the group.

To join a group, a new replica copies a neighbor's group information, and floods a join message to the rest of the group. If a replica does not send any messages for a long time, it is left out when the group topology is recomputed.

Topology update messages carry a sequence number to identify updates and to avoid duplicate messages. When a replica receives a topology update, it floods the update according to the current topology before committing the new topology. Replica update messages also carry a topology sequence number. If a replica learns of an update that carries a higher topology sequence number, it knows that it has the wrong topology and the update has to be re-flooded.

6.2.4 Multicast

Another concept for data replication is the Internet Protocol (IP) multicast [Deering]. IP multicast delivers best-effort datagrams to a group of hosts sharing a single IP multicast address. IP multicasting builds a minimum delay topology to transmit a datagram packet to the group of recipients. It optimizes delivery delay rather than link bandwidth utilization. Currently, IP multicasting is limited to operate within a single routing domain. It uses virtual point-to-point links, or tunnels, to transmit multicast packets between multicast routers in different routing domains. Before transmitting multicast packets through a tunnel, the source multicast router encapsulates them, so that they look like ordinary datagrams to intermediate routers and subnets. The Internet Multicast Backbone (MBONE) [Macedonia and Brutzman] makes use of the multicast concept.

Since, the IP multicast is an unreliable protocol, it does not provide reliable and ordered delivery. This is no problem for real-time audio and video applications, where data loss can be accepted. But, data replication requires reliable service. With an unreliable protocol, data could be lost, which could lead to inconsistency of the replicated services. There are many groups working on a reliable multicast, but unfortunately it is a difficult technology to develop.

Hopwise Reliable Multicast has been suggested as an alternative method to transport replicated data. In this approach the data is sent reliably between network nodes [Donnelley]. A deeper analysis of the method is out of scope of this thesis.

6.2.5 Selecting between replicas

One of the main problems in replication is to select the "best" server from where to fetch the replicated document. The selection should guarantee short response times for the user, but at the same time distribute requests in a fashion that minimizes network traffic, and divides the load evenly between the servers offering the document. Especially, the traffic over regional and long-haul links should be minimized.

It is often the user, who makes the decision which server to use. Sometimes a default server has been defined in the system environment or the client program. Without a default server, the user tends to contact the first appropriate server he knows of. This is often the case for FTP archives and replicated WWW services. This behavior tends to cause unnecessary network traffic and uneven load distribution.

Dynamic server selection

The user should be able to use the most appropriate service site without a prior knowledge of server location or network topology. This could be achieved with a dynamic server selection approach, where an appropriate server would be dynamically selected for the user. Such an approach will be even more motivated by the increasing use of mobile clients, where the choice of the optimal server will depend on the changing location of the client.

A dynamic server selection strategy requires a list of hosts which can provide the given service and a method for quickly estimating the cost of requesting the service from each host. Dynamic server selection has been considered in [Crovella]. Their work indicates that a dynamic server selection mechanism would enable a very flexible and responsive policy for locating servers. It is likely to be more effective than static selection policies, but it is more difficult to implement.

The simplest dynamic selection method would be to map requests randomly or alternately to different servers. The latter approach is used in the NCSA scalable WWW server, which will be presented in Section 6.3. The disadvantage of these simple approaches are that they do not segregate traffic by network region, but they can help to reduce server load problems.

Another method is to probe a set of servers which may have replicas, as well as the replica's home server, and choose the first to reply as source. This approach is used in the Harvest system.

Proper dynamic server selection will require a characterization of the load on a server as well as a measurement of the available bandwidth to each candidate server. In addition, careful study of the time constants and autocorrelation of

server path performance is needed to understand how often path measurements should be conducted.

6.2.6 Locating nearby copies

To effectively select among replicas, nearby servers first have to be located. This problem has been considered by Guyton and Schwartz [Guyton and Schwartz]. They evaluated a variety of server location techniques, uncovering a number of tradeoffs between ease of deployment, effectiveness, network cost and portability. Their work shows that there is no obvious best approach, but only a variety of compromises.

Guyton and Schwartz state that the problem is easy for extreme degrees of replication. If there are only a few replicas, crude approximation, such as mapping domain names to broad geographic regions each containing one replica, can be used. Because network topology often does not correspond to geographic proximity, this approach may not work well. At the opposite extreme, if there are replicas in every subnet, it possible to use broadcast to locate nearby servers having the replica. For cases between these two extremes other approaches have to be used (see Fig. 19).

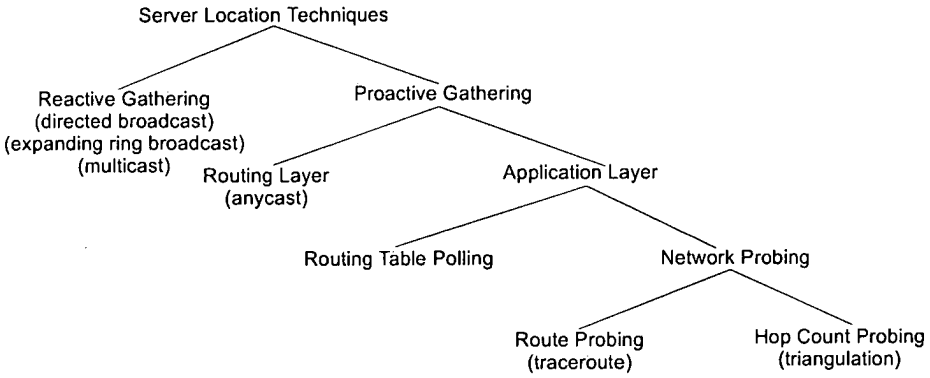

Figure 19: Server location techniques

The server location information can be gathered in reaction to client requests for nearby servers, or it can be gathered in advance. These two ways of gathering the information are called reactive and proactive, respectively.

Reactive gathering includes methods such as directed broadcast, expanding ring broadcast and multicast. In the directed broadcast method a host sends a broadcast message to a particular network and waits for an answer from a server providing the wanted service. The expanding ring broadcast method allows broadcasting to increasingly larger concentric circles around the broadcasting host. These two methods were defined by Boggs [Boggs] and expanding

ring broadcast was later incorporated into the Network Binding Protocol of Xerox [Xerox]. More recent approaches have considered various forms of multicasting to obtain the server location information. When multicasting is used, servers providing the same service join a multicast group. When looking for the nearest server, the one responding most quickly to a group multicast message is chosen.

Instead of gathering the information reactively, it could be gathered in advance. This usually brings an improved performance, since there are no delays caused by gathering the information.

The information could be gathered in the routing layer, reducing gathering costs. Partridge et al. have proposed an anycast mechanism which could be implemented on this level [Partridge et al.]. Contrary to multicast, in anycast only one of the servers belonging to the group is asked if it is ready to provide the service. The information about location has been gathered in advance by network routers. When a new replicated server is created, the local router advertises the existence of the service as a part of its normal routing exchanges with the neighboring routers. Each neighboring router examines a distance metric for the advertised server, updates its location tables, and passes along the updated information to its neighbors. Each router retains knowledge of at least the route to the nearest server, and possibly a small list of alternative servers in case it is notified that the previously nearest server is down. The details of anycast are yet to be worked out.

In contrast to information gathering on the routing-level, a location service on the application-level could allow server quality to be considered. It can be a practical advantage to implement it at the application-level, but this will be at a higher network cost. The main disadvantage of building the server location database at this level is that the database requires periodic updating.

The information could be gathered by polling routing tables or via network polling. The latter could be done with route probing, for example, with the traceroute program. Traceroute counts the number of hosts between the probing host and the destination and the delays between the hosts.

Instead of route probing, hop count probing or triangulation could be used. In this technique, so-called beacon servers, distributed around the network measure their distance to each replicated server. Each beacon server then sends its results to a triangulation server, which combines the measurements to produce a coordinate for each replicated server. When a client wishes to locate a nearby server, each beacon measures its distance to the client, and the triangulation server collects these distances to determine a coordinate for the client. Based on the client's coordinates and the database of replicated server coordinates, the triangulation server then generates a list of servers whose coordinates are closest to that of the client.

6.3 A scalable WWW server

The National Center for Supercomputing Applications (NCSA) has implemented a scalable WWW server [Katz et al.], to meet the demands of the increasing number of users at their WWW site. Load distribution is achieved by alternately mapping the hostname alias of the WWW server to the IP addresses of different WWW servers.

The system (see Fig. 20) consists of a cluster of identically configured WWW servers. When the client makes a request to a WWW server, the server name is resolved at the Domain Name Server (DNS). The NCSA uses a modified DNS[13], which rotates through a list of several IP addresses in round-robin manner. Thus two successive DNS-requests will be mapped to two different WWW servers situated successive in the DNS list.

The documents or files provided by the WWW servers actually reside on several different file servers (Fig. 20). A distributed file system, in this case Andrew File System (AFS), is used to produce a single, coherent view of the file hierarchy. The distributed file system allows the WWW servers to function as if the data was coming from local disk without regard to which file server actually offers the file. The distributed file system uses a local caching scheme which caches all file requests to local disk. Though subsequent data requests for files already stored in the cache are handled locally. The data has also been replicated so that if one of the file servers is unavailable, it will not affect the availability of data to the WWW servers.

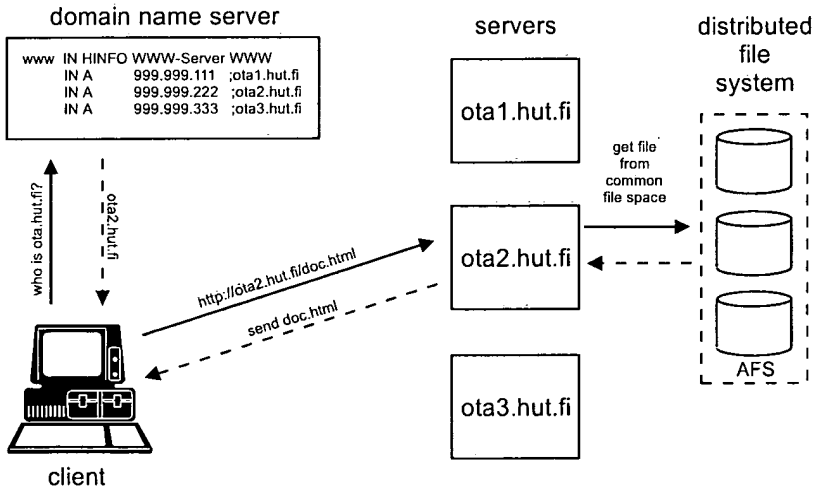

Figure 20: Structure of the NCSA scalable World Wide Web server

[13]NCSA's modified BIND 4.9.2 is available from ftp://nic.ncsa.uiuc.edu/pub/ncsa-bind492.tar.Z

As a result of this design, it is possible to add and remove any number of WWW servers to the pool, dynamically increasing the load capacity of the system. This concept eliminates the vulnerability and single point of failure of a single-server configuration and increases the likelihood for continued and sustained availability.

Drawbacks are that the load is distributed in an unpredictable way, sometimes straining just one of the servers in the cluster. This inequity in distribution makes it difficult to extrapolate growth patterns and predict system vulnerabilities.

The system is also not as dynamic as it first appears. To minimize redundancy and maximize efficiency, the DNS utilizes a caching mechanism to remember data it has already received. The caching causes delays in propagating changes in the DNS hierarchy. The mentioned uneven load distribution is with high certainty caused by this caching mechanism. The IP address of one of the WWW servers will be stored as the equivalent of the WWW server hostname alias, and thus all requests will go to a certain server in the server cluster. The problem can be regulated to some extent by the time-to-live parameter at the Domain Name Server.

6.4 URN mapping

A dynamic lookup method for a client to find the closest proxy server or the closest functional server in a set of servers mirroring each other would offer a flexibility that is now missing in the World Wide Web. It could also be used as a fallback mechanism for proxies so that a client can connect to a second or third proxy server if the primary proxy has failed. It has been suggested [Smith] [Luotonen] [Braun and Claffy] that a scheme similar to the Domain Name Service could be used to implement this [Smith] [Luotonen] [Braun and Claffy].

This scheme places an extra level of mechanism between the end user and the desired information. Instead of specifying document links with host specific URLs, they would be described with globally unique URNs. Each document would then have an entry within a URN server describing the sites where the document can be requested from. When a document is copied to another site, an entry is added to the database at the URN server to indicate this fact.

The URN mapping scheme is presented in Figure 21. When a client makes a request for a document, the local URN server is first contacted. The URN server returns the address of the closest site able to provide the requested document and the document is then requested from that site.

Instead of only returning the address of one site, a list of sites could be returned instead. In this case, it would be up to the client program to decide which of the sites would be the closest one. Closest may reflect metrics including

but not limited to physical distance or number of hops from the query source.

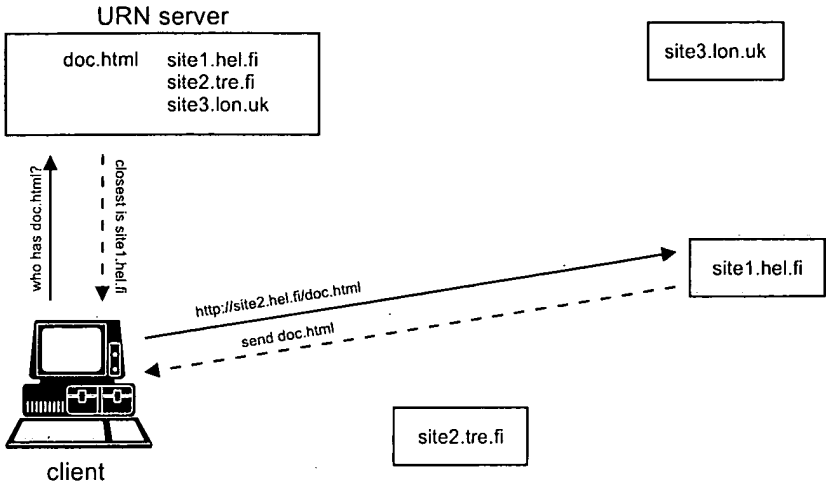

Figure 21: URN mapping

The implementation of URN mapping requires the use of unique URNs. At the moment there is not yet a standard what this URN should look like, but standardization work is in progress. Another open question is if it is really possible to make a single database entry for each document. This seems a bit excessive when taking the number of available World Wide Web documents into account.

6.5 OSF proposal for a scalable fault tolerant WWW server

The OSF Research Institute has announced that it will develop a scalable fault tolerant WWW server [Reynolds] [Goldstein and Dale]. The approach is based on a server cluster and the OSF Mach operating system. The goal of the proposal is to provide uninterrupted service to all clients even though a node fails or is removed for service, and also to allow expansion of the number of nodes in the cluster.

Their proposal is based on developing a toolkit, which will simplify the development of scalable and fault-tolerant applications based on OSF Mach. They achieve fault-tolerance by providing a backup server for each primary server. The backup server is always informed of the state of the primary server and it can take over the processing if notified. The system also has its file system on multiple file servers to guarantee fault-tolerance. File server failures will be handled transparently by switching over to a new file server.

6.6 Speculation

In [Bestavros 95b] Bestavros has considered the possibility of speculation to reduce WWW server load and response time. A speculative service means that when a client requests a document, a number of other documents that the server speculates will be requested by the client in the near future are also sent. The decision is made according to statistical information the server maintains for each document it serves.

Bestavros' work shows that the server load and the service time can be reduced by moderate speculation, without dramatically increasing the requested network bandwidth. Aggressive speculation can be traded for improvements in service times. If clients were cooperative and provided information on what documents they already have, the bandwidth could be better utilized. The benefits of speculation are most pronounced when speculatively served documents are small.

7 The super server concept

As has been seen in the previous chapter, most of the existing scaling methods try to distribute the load among several servers. The problem with these schemes is that the server selection is mainly based on system defaults or on the choice of the user. As observed in Section 6.2.5 this kind of static selection can cause uneven load distribution and unnecessary network load. Instead, it would be better if an appropriate server were dynamically selected for the user.

Dynamic server selection is implemented in the scalable WWW server approach by the NCSA. The method of alternately mapping the requests to different servers in a local cluster might solve local load problems, but it cannot be used to effectively distribute the load in a widely distributed system. The suggested URN mapping, is a better approach for a widely distributed application. However, it might not be efficient to base the choice of server only on geographical information.

A scheme where the requests are directed to an appropriate server according to the location of the requested data, the current load of the servers, the location of the servers, and the available network bandwidth, might bring improvements to the above mentioned schemes. This kind of approach could better adapt itself to the current state of the system.

Such an approach called the *super server concept* will be presented and evaluated in this chapter. Possible system architectures and parameters for the mapping decision will be considered.

7.1 Description of the super server concept

The super server concept is based on a group of servers distributed in the network. The servers are modified World Wide Web servers with caching ability. In the previously presented URN mapping the client contacts a specific server providing the decision of the most appropriate server. In the super server concept the user would contact the modified WWW server as a normal WWW server, and it would then make this decision. If the requested document is not found on the server, the server has high load, or a closer server could be used, the request will be redirected to a more suitable server. The client will use this new server for future requests, as long as the server has the requested document and it has not too high a load.

This scheme will distribute the users among all the available servers. If the location of the users are taken into account, the users will additionally be directed to the closest available server having the document. As long as the load is low the users will be using the closest server. When the load increases some of the

users will be transferred to another server. Thus all users are guaranteed service.

This concept enables servers taking part in the service to provide different sets of documents and different services. This is a good, since servers might have different storage capacities or be specialized in some specific service.

The performance of the concept can be improved through caching. When a server does not have the document and has a low load, the document can be retrieved from another server and be cached. This will help distribute the load, especially if the caching scheme is constructed so that it will favor popular documents. In the same manner, a server having problems with a high load, can replicate a document to another server and redirect clients to it.

These schemes suit only static and non-changing dynamic data. If a server providing changing dynamic data cannot sustain all requests, new distribution points could be activated. The new distribution point will distribute material from the same source. For example, if a database search engine cannot manage all requests, the service could be activated at another server. Both will, however, still use the same database for their searches.

To decide where to redirect, the location of the requested documents and information about server loads and locations must be available. The processing and gathering of this system information can be done in a distributed manner at all servers or in a centralized manner at some specified servers. In the section below I will first describe how the needed information could be gathered and then how the centralized and distributed concept might work.

7.1.1 Location of the documents

To be able to redirect when a document is not found, the load is too high or the client is not using the most appropriate server, the server must know which other servers have the document. There are optional schemes for obtaining this information:

1. The server asks all other servers for the requested document each time the information is needed. The server does not maintain a list of the locations of the documents.

2. After asking all servers for a document, the location information is stored. The information will be used a certain period of time for other redirects. After a certain time, the information is removed or renewed by asking all servers if they still have the document.

3. All servers manage a document list. Whenever a server adds or removes a document, all servers are informed of the change and the lists are updated.

To be able to locate the document, its name must be the same at all servers. If all serves would have the same documents, no location information would be needed. If the servers have a different set of documents the above mentioned options are to be considered.

If the documents change location more often than the location information is needed, option one causes less network traffic than option three. When the documents change location more seldom option three is better. Option two causes less network traffic than option one, but in case of frequent changes in document locations, the method can return stale location information. If documents change location infrequently the method might be as effective as option three.

7.1.2 The load of the servers

To be able to decide whether to redirect or not, the server must monitor its own load. This can be done with a load monitor program or be estimated by how many requests are processed per second.

When deciding where to redirect, the load information of other servers must be available. There are different ways in which this information can be gathered and used.

1. The servers tell each other periodically what their current load is.
2. When a server reaches a certain load level it informs all other servers that it does not want any more requests redirected to it. When the situation eases, the server invalidates the restriction.
3. The load information of other servers is not known when redirecting.

It is not optimal to tell all the other servers the load periodically, since the actual load might vary during the period between the load messages. To describe the actual situation, load information should be sent almost continuously. This would increase the network traffic significantly. A more reasonable approach is to tell the other servers when no more requests are wanted.

If the load threshold is based only on a certain load level, the server might start oscillating between sending "don't redirect" and "redirect". To avoid this, several threshold levels should be defined. When a certain load is reached, the server will send a "try to avoid me" message. When the load increases over the definite load threshold, it sends a "don't redirect here" message.

If the servers redirect without using load information from other servers, the client might be redirected several times. If a client is redirected to a server also having too high a load, it will be redirected again. Thus, redirection without using load information from other servers is not a very good approach.

7.1.3 Location of the servers

The information of the best situated server in the network topology is needed for two different tasks. The server needs the information to decide whether or not to serve the request and to be able to redirect the request to a more appropriate server.

The meaning of "best situated server in the network topology" might vary in different server selection policies. The aim can be to keep the network traffic local, and avoid unnecessary traffic over regional and long-haul links. Alternatively, the aim can be to minimize the network latency for the client. Often, serving the client locally leads to minimal latency. However, this is not always the case, since it depends on the available network bandwidth.

One method to approximate the optimal server would be to use geographical information to assign network domains to different servers. This is not optimal, since network topology does not often correspond to geographic proximity and the network bandwidth is not taken into account. The question is also what to do when there is a request from a domain that has not been appointed to any server.

Another method is to determine the best situated server according to probed round-trip times. When a server receives a request, it will order all servers to probe the round-trip to the client. Based on the results received from the other servers and the own round-trip measurement, the server will make the decision of the best situated server. Since it is a bit excessive to gather the information for every document request, the results could be saved for a certain period of time.

The two methods above could be combined. Some specific domains will be appointed to each of the servers. When the server receives a request from an unknown domain, the other servers are asked to probe the client. According to the results, the whole domain in which the client resides will be mapped to the server with the shortest round-trip time. To base the decision on only one probe is not optimal, since round-trip times usually vary.

7.1.4 Distributed versus centralized super server architecture

There are two optional ways how the processing and gathering of the location and system information could be done: in a distributed manner at all servers, or in a centralized manner at some specific servers.

In the distributed super server architecture, presented in Figure 22, each server participating in the service gathers location and system information. To distribute the information, all servers must communicate with each other. For this a specific communication protocol is used.

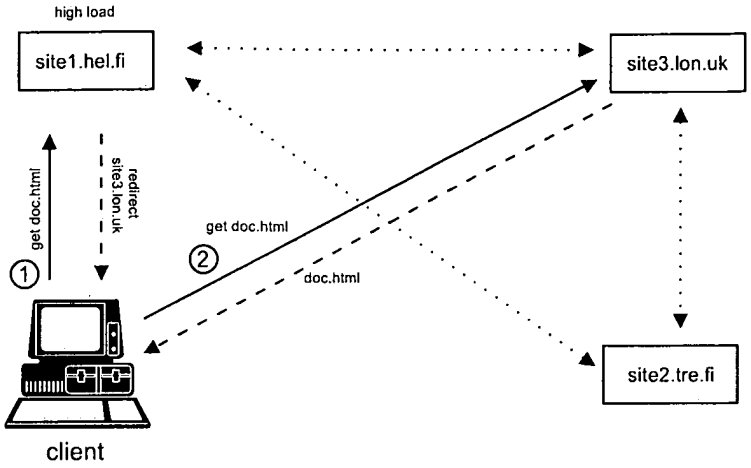

Figure 22: Distributed super server architecture

In the centralized super server architecture, presented in Figure 23, there are two kinds of servers participating in the service: normal servers, which serve the clients, and super servers, which gather location and system information. There are one or more super servers depending on the size of the service. The ordinary servers monitor their own load and they know which clients to accept. If a request needs to be redirected, the information for making this decision is asked from the super server.

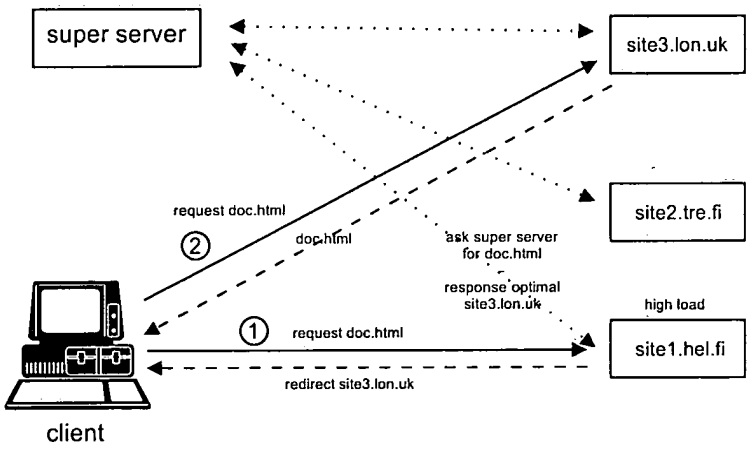

Figure 23: Centralized super server architecture

The distributed architecture might cause a significant amount of network traffic, since all servers need to gather the location and system information. All servers are also strained with the processing and gathering of the information.

The centralized version is likely to cause less network traffic, but on the other

hand, the super server can be a crucial point. If it fails, the whole system fails. To avoid this, there should be one or more backup servers for the super server.

8 Testing the super server concept

I have implemented the super server concept to be able to determine its efficiency. The goal was to see how a system based on the ideas in the previous chapter might work, and to indicate crucial points in the system architecture.

8.1 Implementation of the super server

The super server was implemented as a WWW server. It is written in Tcl[14]. In addition to the basic Tcl, the extensions Tclx and Tcl-dp were used. Some additional network and inter-process communication functions were also added to Tcl. The server comprises the most important functions of a WWW server. It can serve documents and it is capable of spawning CGI scripts, including imagemaps. Imagemaps are images where different regions contain hyperlinks to other data objects.

The server is constructed as a pre-forking server with caching ability. To implement these features, the server includes three types of modules: a relay, a cache and httpd frontends. Figure 24 describes the server modules and their internal relationship. The relay is the co-ordinator of the server. It manages all processes and functions. The httpd frontends process user requests and the cache module maintains the cache.

When the server is started, the relay forks a number of httpd frontends. It assigns one of them to listen at the server port. All other httpd frontends will wait idly. When the listening httpd frontend receives a request, it informs the relay about the event. The relay then selects one of the other httpd frontends to listen at the server port. If there is no idle httpd frontend available, the relay forks new ones. The default is to fork four httpd frontends.

The caching module keeps track of the cached documents. If the cache space becomes full, the least used documents will be removed. Caching is used only when documents do not reside at all servers.

The server can be run in three different modes: standalone, document distribution, and load checking mode. In the standalone mode the server works as a normal World Wide Web server. When run in the two other modes the server works as a modified WWW server according to the distributed super server concept. The server is also already partly prepared for a centralized super server concept. The implementation of the different modes will be described in the following sections.

[14]Tcl WWW info <URL:http://www.sco.com/Technology/tcl/Tcl.html>

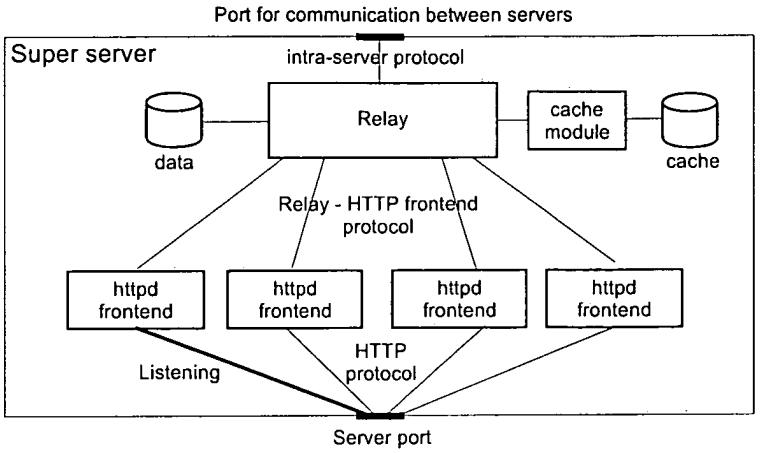

Figure 24: The server modules and their internal relationship

8.1.1 Standalone mode

In the standalone mode, the server works as a normal WWW server. Upon a user request one of the httpd frontends sends the requested document to the client. If the document is not found or an error occurs, an error messsage is sent. The relay manages the number and state of the httpd frontends. In standalone mode, the caching module and redirection of requests are not used.

8.1.2 Document distribution mode

When used in one of the two distributed modes, the httpd frontend will ask the relay how to handle the received user request. The relay decides regarding to the server load whether the request is served or redirected. The load is monitored by counting the number of httpd frontends serving clients.

When sending an HTML document to the client, the httpd frontend will look for hyperlinks to images in the document. For each URL to an image it will ask the relay for the most appropriate server which could provide the image. The URLs to images are then modified according to the answers from the relay. In this manner, the client will receive a document with optimized URLs for the images. The distribution of the load is optimized by using two load thresholds for deciding when to redirect. When the lower load threshold is exceeded the URLs of the images are replaced with URLs to other servers. When the higher limit is exceeded all requests are redirected with the HTML "Redirect" message.

In the distribution mode, documents do not have to reside at each server. Documents will be replicated to other servers when the document is needed and the server load allows it to be replicated. In this mode, the servers also maintain

a document location list. The information in the list is gathered successively.

Client requests are handled as follows: If the load is acceptable and the relay finds the document at the server, the httpd frontend is told to send the document to the client. If the document is not found and the load is acceptable, the server will try to obtain the document from another server and cache it. If the load is too high, the request will be redirected to another server.

In order to know where to redirect, the server maintains a list of document locations. If a server does not have a document, it checks for an entry for the document in the location list. If an entry is found, it asks all servers having the document. The fastest answering server will be selected as source. When the document cannot be found in the mentioned list, the relay will initiate a query to all relays. This will always be done if no answers are received on the queries made to the locations which are supposed to have the document according to the internal document list. In this case, too, the fastest server will be chosen, and all answers will be stored in the document location list.

If a server receives an inquiry for a document from another server, it first checks its own load. If it is smaller than the threshold, it will send an answer to the server, otherwise it will wait a configurable time and send a delayed response. If the document cannot be found the server will do nothing.

8.1.3 Load checking mode

In the load checking mode it is assumed that all servers have all documents. Thus, no asking needs to be done and no document list needs to be managed. Different from the distribution mode, the servers are configured to periodically tell each other their current load. Additionally, the servers are defined as high- or low-performance servers, according to the characteristics of the host and the network. The clients will first be distributed among the high-performance servers. Clients will only be redirected to low-performance servers, when all high-performance servers have a high load.

In the load checking mode, all servers measure their maximum load during a five second period and send the result to all other servers periodically. No load message is sent if the load has not changed. The load values received from other servers are stored.

Each server will memorize the high-performance server with the lowest load. To achieve load-balancing, a server will redirect a client to this selected high-performance server when its own load is significantly higher than the load of the high-performance server. In this way, the load is equally balanced among the high-performance servers. A redirect to a low-performance server will only occur when no high-performance server with lower load is found.

8.2 Conducted tests

To determine the efficiency of the super server concept, the server was tested in all three presented modes. The server was first tested in standalone mode and then in both distributed modes.

8.2.1 Test environment

The tests were conducted in the local network of the Helsinki University of Technology. In the standalone test the server was started on the host max.hut.fi. In the distributed tests three additional servers were started on the hosts lk-hp-24.hut.fi, lk-hp-33.hut.fi and lk-hp-34.hut.fi. The host max.hut.fi is a DEC Alpha 1000 server with 128 MB memory. The other hosts are HP 705 workstations with 64 MB memory. Figure 25 shows the locations of the hosts in the network.

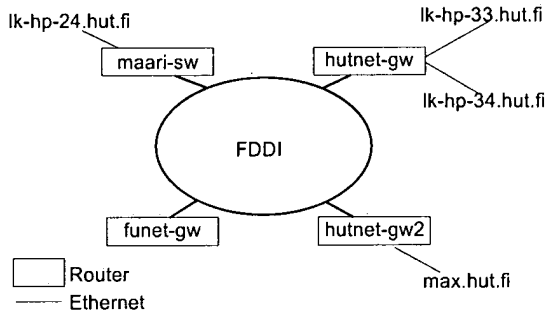

Figure 25: The location of the test hosts in the local network of the Helsinki University of Technology

All servers were configured to use a local disk for storing and caching the data. The data to be distributed was copied from the OtaOnline service.

The load was created with the same robot program as in the OtaOnline load test (Section 5.6.3). All robots were configured to first fetch the same HTML document, the front page of the OtaOnline service, from max.hut.fi. After that they randomly followed a link in the received document, making a random pause of 0-60 seconds between the requests.

In all tests, a total number of 80 robots were successively started on four hosts in the hut.fi domain and on four hosts in the cs.hut.fi domain. Ten robots were started on each host.

8.2.2 Standalone mode results

To be able to compare the efficiency of the distributed approaches to the performance of a single server, the server was first tested in standalone mode on max.hut.fi.

The results of the test are shown in Figure 26. When the robots were successively started, the number of connections first increased. After a while the increase stopped, even though new robots were started. Instead, timeouts started to occur. The variation in the number of connections is due to the random request rate of the robots.

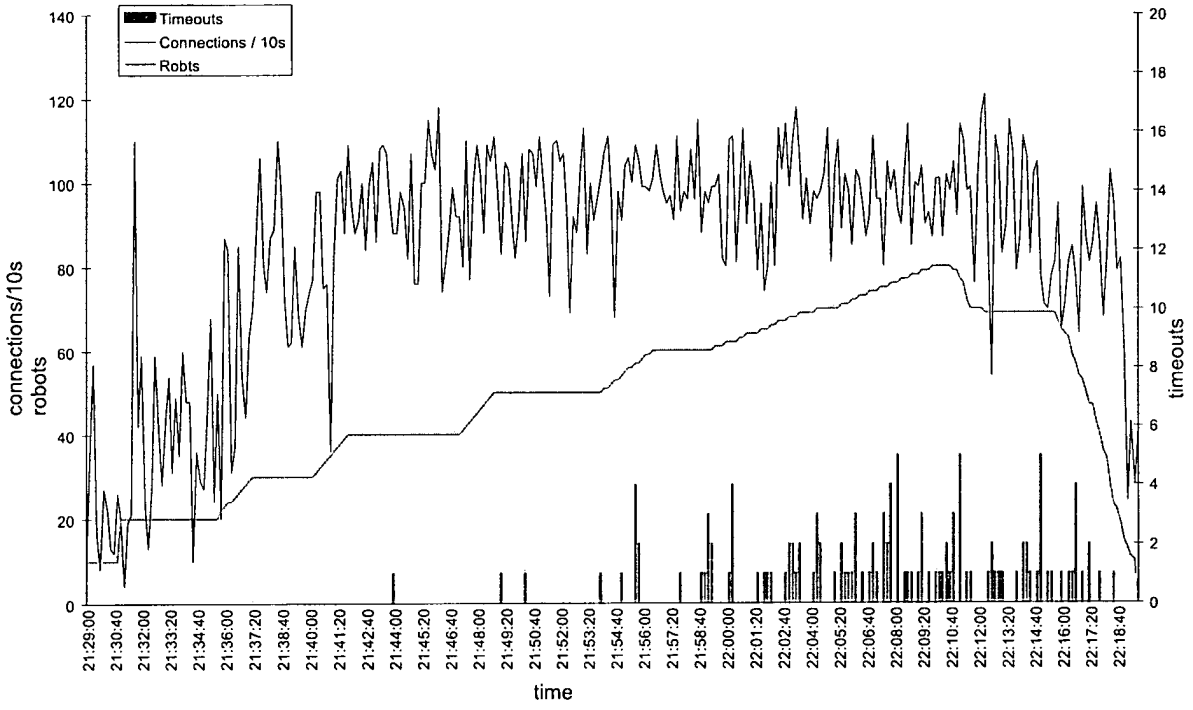

Figure 26: The number of connections, robots and timeouts as function of time for the standalone server

The timeouts were probably caused by the fact that the server was not able to process all incoming requests. The server is only able to handle a certain number of requests per second. If a higher rate is experienced, requests are put in a queue. If the queue cannot be emptied fast enough, timeouts will occur.

The test indicates that the maximum performance of the standalone server in this specific test environment is about 95 connections per 10 seconds.

8.2.3 Document distribution mode results

The test was conducted by starting a server on each of the hosts max.hut.fi, lk-hp-24.hut.fi, lk-hp-33.hut.fi and lk-hp-34.hut.fi. The host max.hut.fi was defined to have all the documents of the service. The other hosts were defined to have none. All robots were configured to make the first request to max.hut.fi.

In the test max.hut.fi was expected to redirect users and to replicate documents to other servers. The results were not reassuring.

The test results showed that the communication between the servers is a severe bottleneck. It can cause severe reductions in the overall system performance. As described before, a server will ask many if not all servers each time it wants to redirect, causing a lot of network traffic and processing overhead in the servers.

The test also showed that a scheme where the documents are replicated when they are requested the first time by a client, does not work if documents have to be replicated in a phase of high load due to many requests. At the beginning, max.hut.fi was the only server having the documents. When max.hut.fi redirected requests to another server, it also had to transfer the requested document to the server. This only increased the load of max.hut.fi. This means that during a certain period when documents were distributed among all servers, the overall performance was very poor, not to say unacceptable.

8.2.4 Load check mode results

As in the previous test, servers were started on the hosts max.hut.fi, lk-hp-24.hut.fi, lk-hp-33.hut.fi and lk-hp-34.hut.fi. All servers had all the documents of the service. The servers on the hosts max.hut.fi and lk-hp-24.hut.fi were classified as high-performance servers. The robots were configured to request the first document from max.hut.fi. In the test the requests were expected to be distributed evenly among the servers.

The results of the test are presented in Figure 27. In the figure, the connection rate of each server and the number of robots are presented as a function of time.

In the test, the requests were first distributed to both high-performance servers. Since lk-hp-24.hut.fi is not as powerful a computer as max.hut.fi, it soon redirected to the low-performance servers on lk-hp-34.hut.fi and lk-hp-33.hut.fi. Thus the load was distributed among all servers, as seen in Figure 27. The server on max.hut.fi handles more requests than the other servers, since it is more powerful.

Figure 28 gives a more detailed view of the request rates. The number of requests in 10 seconds was divided by the absolute peak value of requests to enable a better comparison of the connection rates.

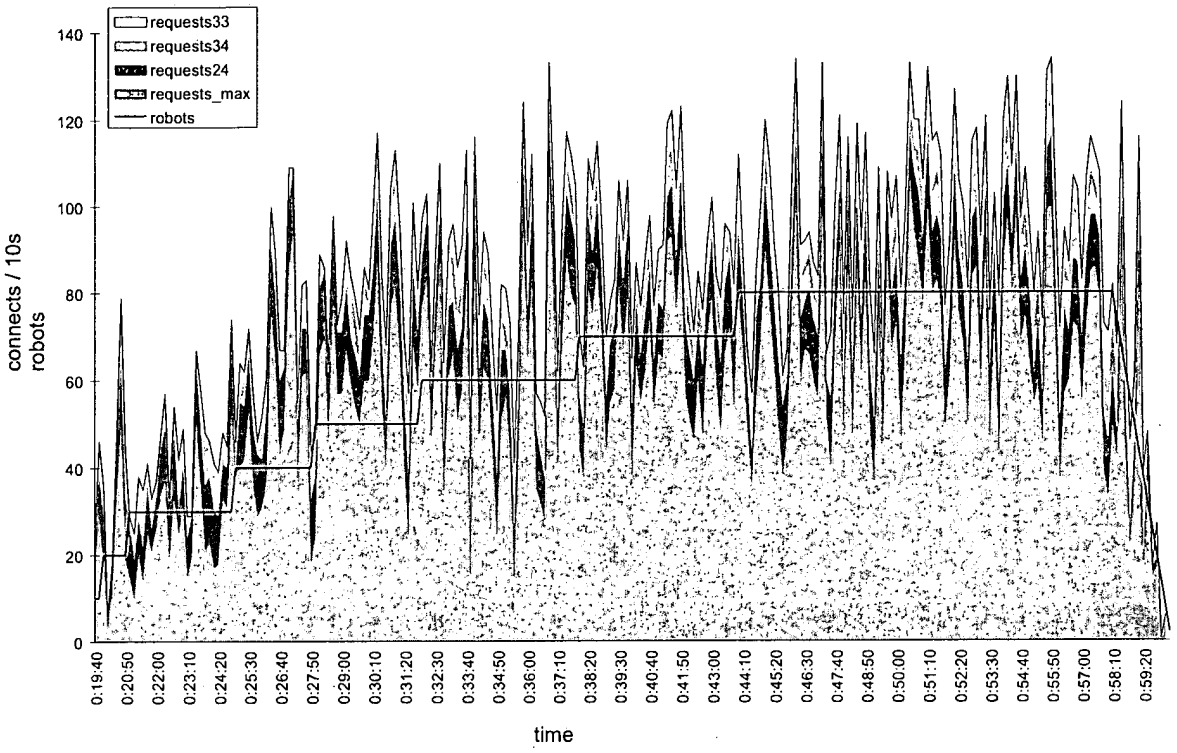

Figure 27: Number of connections per 10 seconds and the number of robots when the servers were run in load check mode

As can be seen in Figure 28, the number of connections increases at max.hut.fi at 0:23:20. After that the connection rate of the second high-performance server, lk-hp-24.hut.fi, increases too. This is because max.hut.fi has redirected some of the incoming requests to the second high-performance server. At the same time, lk-hp-24.hut.fi begins to redirect some clients to the low-performance server lk-hp-34.hut.fi, whose load also increases. Ten seconds later it will reach its load threshold and begin to redirect to the second low-performance-server, lk-hp-33.hut.fi. As a result of this, within 20 seconds the load of the two high-performance servers reaches again a normal level. After the request rate has settled down again, the request rate at the second low-performance server, lk-hp-33.hut.fi will decrease, too.

From test trials it is clear that the load-measurement method and the definition of the load thresholds have a great influence on the efficiency of the system. In this test the load thresholds of the servers were configured based on empirical tests of how many connections a machine could serve without creating timeouts.

In the test the requests were distributed on the four servers, even though all

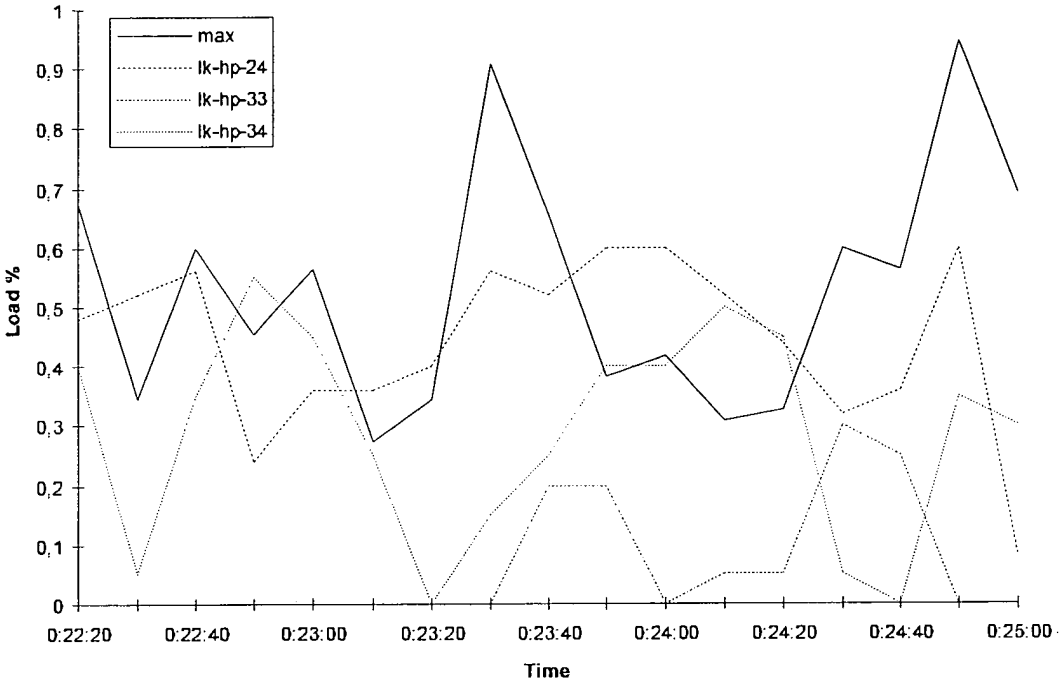

Figure 28: The distribution of served request on each of the test servers

initial requests were done to max.hut.fi. The system reached a slightly higher connection rate as a server in standalone mode. The system could have managed more requests, if the load thresholds had been defined differently. The three HP 705 workstations, i.e. the hosts lk-hp-24.hut.fi, lk-hp-33.hut.fi and lk-hp-34.hut.fi, are less powerful than a DEC Alpha 1000 server, i.e. max.hut.fi. The thresholds for the second high-performance server, lk-hp-24, were set too high, which caused an overload on this server. This reduced the overall performance of the whole distributed server system. The fact that all timeouts in this experiment were caused by this specific server supports this assumption.

8.3 Summary of the test results

The tests were performed to give an idea how a distributed system, based on the ideas presented in the previous chapter, might work. The tests were only expected to give qualitative results.

The performed tests indicate that the communication between the servers is a critical factor when designing a distributed system. The communication should

be carefully planned and unnecessary messaging should be avoided.

How the server load is measured is also a crucial factor for the overall performance of a dynamic distributed system. The load should reflect the real situation of the server. The load thresholds are also difficult to set, since the acceptable values may vary according to the state of the system.

The tests also show that the distribution of documents should be carefully planned. If many or large documents have to be distributed to multiple hosts during a short period of time performance problems might appear.

As a summary it can be established that it is difficult to design a well working dynamic distributed system. Such a system requires thorough planning and testing. Many important experiences can certainly be gained by studying existing distributed systems.

9 Proposal of a scalable server architecture

Based on the previous chapters, I will propose a scalable server architecture. The proposal is based on properties which an interactive news system, such as OtaOnline, is considered to require.

The OtaOnline news system will provide interactive multimedia news and information. The provided documents will consist of data objects in different media forms: text, images, video, and audio. In addition to data which is more or less frequently updated, the service will provide real-time data, as well as access to old data stored in databases. Thus, the system will distribute all types of data, i.e. static, non-changing and changing dynamic data. In addition to different types of data, the system will include mechanisms for offering personalized views of the information. The structure of the documents will be described in a document description language. The aim is to be able to offer the information in different data formats. With help of metadata, the documents are created and converted to the wanted format, e.g. HTML. The personalization will also require similar processing for the creation of personalized versions. The system is also planned to gather information to be used for regulating the system. In addition to all these requirements, the system should be scalable.

Because of the above described features, OtaOnline cannot be compared to current World Wide Web services. Providing documents which are created first when they are requested will cause higher load, than the providing of static documents. To guarantee a good performance to all users, the service has to be distributed between strategic located servers in the network. The existing replication and caching schemes will not be sufficient. It would be inefficient to cache or replicate dynamically generated data, which might be different the next time it is accessed. This can especially happen for personalized data. Current replication schemes are not optimized to distribute the load evenly among different servers, either.

A system where users are distributed dynamically to the most appropriate server, according to the current server loads and the available network bandwidth, might be a good approach for OtaOnline. With such an approach the system load can be better distributed. Another justification for such an architecture is the fact that the system might include services which are located on some specific server. In this case, there must be a mechanism for directing users to the server having the wanted service.

Based on the considered requirements, a proposal for a scalable system architecture for the OtaOnline news system is presented in Figure 29. All servers include the components described in server B.

The figure shows an example of four servers connected in a distributed man-

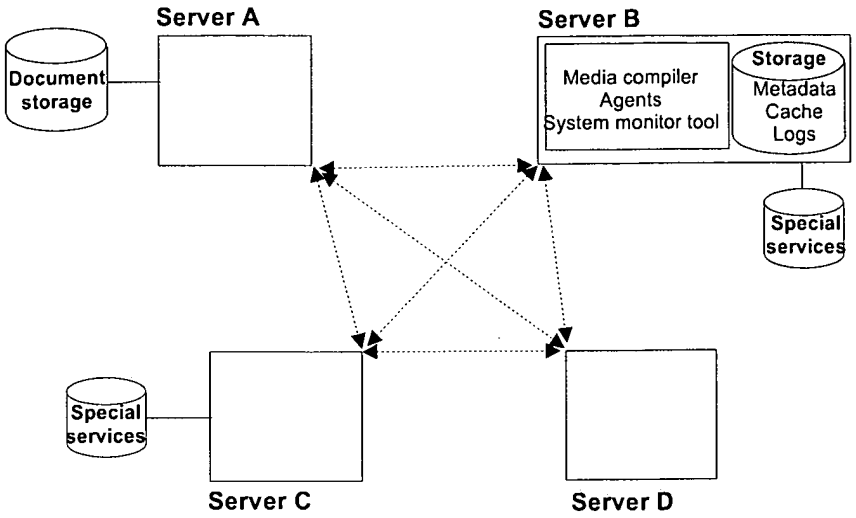

Figure 29: A proposal of a scalable system architecture for the OtaOnline news system. All servers include the components described in server B

ner. For communication between servers, each server maintains communication links to all other servers in the system. The links are used to exchange system information, such as load.

Each server must have effective means to monitor its own load. The decisions of what server the user should use will be based on the monitored system load. Load balancing techniques, described in the load-check-mode test (Section 8.1.3), could be used.

In the system some documents will be created dynamically and others will be static. The static documents can be directly distributed to the servers. There is no sense in replicating dynamic data.

All data objects and metadata describing the documents will reside in a centralized repository. One of the servers is assigned to handle the data distribution. In the example it is server A. The server will not be used for serving clients, since this caused severe performance problems in the document-distribution-mode test described in Section 8.2.3. The distribution of data will be done through the server-communication links. The metadata will be distributed to all servers. The static documents could be distributed to the servers according to measured and predicted request rates.

The metadata is distributed to enable the generation of the documents on each server. In this manner the load caused by generating documents are distributed to all servers. The generation of the documents will be done by a media compiler residing on each server. An invalidation scheme will be needed to inform the servers when new data is inserted in the system.

To be able to offer personalized data the servers include agents. User profiles describing the preferences of the users will be stored in a centralized manner on one server. Whenever a new reading session is started, the user profile is distributed to the server which the user uses. An invalidation scheme will be needed to notify all servers about changes in the user profiles.

To be able to monitor the system, each server includes a system monitor tool. In addition to monitoring the system state, e.g. load, it is used for gathering statistics about the interest in different documents and information about the users. This information can be used to distribute the data more effectively and to better direct users to appropriate servers.

As mentioned above, the system can include special services. In Figure 29, the servers B and C both include a special service. All other servers need to know the location of the special services to be able to direct requests correctly.

To make the system more flexible, a scheme for dynamically adding and removing servers has to be considered. A new server could join by contacting the dedicated document server, and then receive a list of all other participating servers. To locate the document server, different location techniques described in Section 6.2.6 could be used. The newcomer will set up communication links to all the other servers and exchange necessary status information.

As seen in the previous chapter, it is challenging to design a distributed system. To implement this proposed system careful consideration will be required. The system should be analyzed thoroughly for possible performance bottlenecks. It might prove profitable to first concentrate on a few parameters in the design. This means to start with a simple system and successively include further optimizations.

10 Summary

The interest in and the use of distributed multimedia systems have increased rapidly. One reason for this is the wide implementation potential such systems offer. Distributed multimedia systems enable clients to access remote services, and to freely choose what data to receive. The client is able to personally control the manner and time of receipt of the data.

The progress of distributed multimedia systems has been feasible, because of advances in network and computer technology. However, there are still many technical challenges, since multimedia data differs significantly from traditional data that current computer systems are designed to handle.

Scalability is one of the most important factors when designing distributed multimedia systems which will provide interactive services for a wide clientele. The system must be able to adapt itself to different numbers of users and amounts of data, without resource problems or performance bottlenecks. Scalability might be the crucial factor for the success or failure of a service.

Scalability problems can be caused by the growth of the user base and amount of data, as well as by the increase in size of the data objects and non-uniform request distribution. To get the system to scale better, approaches such as caching and replication of data, dynamic request mapping, regulation of the quality and price of service, and analysis of access patterns can be used. The system architecture, and the characteristics of the provided service and data affect what kind of scaling methods can be used. In the World Wide Web the most popular scaling methods have been different caching and replication schemes.

Most of the existing scaling methods try to distribute the load by distributing users to several servers. However, the selection of what server to use is mainly based on system defaults or the choice of the user. This can cause uneven load distribution and obstruct the scalability of a system. Schemes where the users are directed to an appropriate server according to the the location of the requested document, the current load of the servers, the location of the servers, and the available network bandwidth, might bring an improvement.

A system based on such schemes was considered and analyzed. The concept was tested for different design possibilities. The performed tests indicate that the communication between servers, the load measurement and distribution of documents are factors that have to be considered carefully. Such a system requires a thorough planning and testing.

A system where users are dynamically directed to an appropriate server might well be implemented for an interactive news system like OtaOnline. If such a system is to be implemented, it might prove profitable to first concentrate on a few parameters in the design. This means to start with a simple system and

successively include further optimizations.

References

[Berners-Lee et al. 94]
Berners-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H. F., and Secret, A. 1994. *The World-Wide Web.* Communications of the ACM, volume 37, number 8. Pages 98-107.

[Berners-Lee et al. 95]
Berners-Lee, T., Fielding, R. T., and Frystyk Nielsen, H. 1995. *Hypertext Transfer Protocol – HTTP/1.0.* Internet draft, March 8, 1995. http://www.ics.uci.edu/pub/ietf/http/draft-ietf-http-v10-spec-00.txt

[Berra et al.]
Berra, P. B., Chen, C-Y., Ghafoor, A., and Little, T. D. C. 1992. *Issues in Networking and Data Management of Distributed Multimedia Systems.* Symposium on High-Performance Distributed Computing, Syracuse, New York. 18 pages.

[Bestavros et al.]
Bestavros, A., Carter, R. L., Crovella, M. E., Cunha, C. R., Heddaya A., and Mirdad, S. A. 1995. *Application-Level Document Caching in the Internet.* Technical Report BU-CS-95-002, Computer Science Department, Boston University, January 15, 1995 (Revised March 23, 1995). 19 pages.

[Bestavros 95a]
Bestavros, A. 1995. *Demand-based Document Dissemination for the World-Wide Web.* Technical Report TR-95-003, Computer Science Department, Boston University, February 15, 1995. 22 pages.

[Bestavros 95b]
Bestavros, A. 1995. *Using Speculation to Reduce Server Load and Service Time on the WWW.* Technical Report TR-95-006, Computer Science Department, Boston University, February 21, 1995. 15 pages.

[Blaze]
Blaze, M. A. 1993. *Caching in Large-scale Distributed File Systems.* Technical Report TR-397-92, Princeton University, January 1993. 90 pages.

[Boggs]
Boggs, D. R. 1983. *Internet broadcasting*, Ph.D thesis, Technical Report CSL-83-3, Xerox Palo Alto Research Center, October 1983.

[Borenstein and Freed]
Borenstein, N., and Freed, N. 1993. *MIME (Multipurpose Internet Mail*

Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies. Internet Request for Comments RFC 1521, September 1993. 81 pages.

[Bowman et al. 94a]
Bowman, C. M., Danzig, P. B., Hardy, D. R., Manber, U., and Schwartz, M. F. 1994. *The Harvest Information Discovery and Access System.* The Second International WWW Conference '94, Chicago 17-20 October 1994, Mosaic and the Web, Advanced Proceedings, volume 2. Pages 763-771. ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/Harvest.Conf.ps.Z

[Bowman et al. 94b]
Bowman, C. M., Danzig, P. B., Hardy, D. R., Manber, U., Schwartz, M. F., and Wessels, D. P. 1994. *Harvest: A Scalable, Customizable Discovery and Access System.* Technical Report CU-CS-732-94, Department of Computer Science, University of Colorado, Boulder, August 1994 (Revised March 1995). 29 pages. ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/Harvest.Jour.ps.gz

[Bowman et al. 94c]
Bowman, C. M., Danzig, P. B., Manber, U., and Schwartz, M. E. 1994. *Scalable Internet resource discovery: Research problems and approaches.* Communications of the ACM, volume 37, number 8. Pages 98-107.

[Braun and Claffy]
Braun, H., and Claffy, K. 1994. *Web traffic characterization: an assessment of the impact of caching documents from the NCSA's web server.* The Second International WWW Conference '94, Chicago 17-20 October 1994, Mosaic and the Web, Advanced Proceedings, volume 2. Pages 1007-1027.

[CGI]
CGI: Common Gateway Interface,
http://hoohoo.ncsa.uiuc.edu/cgi/intro.html

[Chankhunthood et al.]
Chankhunthod, A., Danzig, P. B., Neerdaels, C., Schwartz, M. F., and Worrell, K. J. 1995. *A Hierarchical Internet Object Cache.* Technical Report 95-611, Computer Science Department, University of Southern California, Los Angeles, California, March 1995. Also Technical Report CU-CS-766-95, Department of Computer Science, University of Colorado, Boulder, Colorado. 13 pages. ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/HarvestCache.ps.Z

[Crovella and Carter]
Crovella, M. E., and Carter, R. L. 1995. *Dynamic Server Selection in*

the Internet. Technical Report TR-95-014, Computer Science Department, Boston University, June 30, 1995. 5 pages.

[Danzig et al. 93]
Danzig, P. B., Hall, R. S., and Schwartz, M. F. 1993. *A Case for Caching File Objects Inside Internetworks.* Technical Report CU-CS-642-93, Department of Computer Science, University of Colorado at Boulder, March 1993. 15 pages.

[Danzig et al. 94]
Danzig, P. B., DeLucia, D., and Obraczka, K. 1994. *Massively Replicating Services in Wide-Area Internetworks.* Computer Science Department, University of Southern California, January 1994. 13 pages.

[Deering]
Deering, S. 1989. *Host extensions for IP multicasting,* Internet Request for Comments RFC 1112, August, 1989. 17 pages.

[Dŏganata and Tantawi]
Dŏganata, Y. N., and Tantawi, A. N. 1994. *Making a Cost-Effective Video Server.* IEEE Multimedia, volume 1, number 4. Pages 22-30.

[Donnelley]
Donnelley, J. E. 1995. *WWW media distribution via Hopwise Reliable Multicast.* Computer networks and ISDN systems: Proceedings of the Third International World-Wide Web Conference, volume 27, number 6, Elsevier. Pages 781-788.

[Fuhrt]
Fuhrt, B. 1994. *Multimedia Systems: An Overview.* IEEE Multimedia, volume 1, number 1. Pages 47-59.

[Gibbs and Tsichritzis]
Gibbs, S. J., and Tsichritzis, D. C. 1995. *Multimedia Programming - Objects, Environments and Framework,* ACM Press, Addison-Wesley, Padstow. 323 pages.

[Glassman]
Glassman, S. 1994. *A Caching Relay for the World Wide Web.* Computer Networks and ISDN systems, First International Conference on the World-Wide Web, Elsevier Science BV. 8 pages. http://www1.cern.ch/PapersWWW94/steveg.ps

[Goldstein and Dale]
Goldstein, I., and Dale, P. 1994. *A Scalable, Fault Resilient Server for the*

World Wide Web. A proposal by the Research Institute of OSF, October 3, 1994. http://riwww.osf.org:8001/www/interop_prog_book/

[Guyton and Schwartz]
Guyton, J. D., and Schwartz, M. F. 1995. *Locating Nearby Copies of Replicated Internet Servers.* Technical Report CU-CS-762-95, Department of Computer Science, University of Colorado - Boulder, February 1995. 18 pages.

[Gwertzman]
Gwertzman, J. 1995. *Autonomous Replication in Wide-Area Internetworks.* BA thesis, Harvard College, Cambridge, Massachusetts. 95 pages.

[Hautaniemi]
Hautaniemi, M. 1994. *Management and Control of the TCP/IP-network of the Computing Centre at the Helsinki University of Technology,* Master's thesis (in finnish), Helsinki University of Technology, March 1994. 87 pages

[Hoffman]
Hoffman, P. E. 1995. *WWW Servers Comparison Chart*
http://www.proper.com/www/servers-chart.html

[Jadav and Choudhary]
Jadav, D., and Choudhary, A. 1995. *Design Issues in High Performance Media-on-Demand Servers.* Department of Electrical and Computer Engineering and CASE Center, Syracuse University, Syracuse. 18 pages.

[Kantor and Lapsley]
Kantor, B., and Lapsley, P. 1986. *Network News Transfer Protocol - A Proposed Standard for the Streambased Transmission of News.* Internet Request for Comments RFC 977, February 1986. 27 pages.

[Katz et al.]
Katz, E. D., Butler, M., and McGrath, R. A. 1994. *A Scalable HTTP Server: The NCSA Prototype.* Computer Networks and ISDN systems, First International Conference on the World-Wide Web, Elsevier Science BV. 10 pages. http://www1.cern.ch/PapersWWW94/ekatz.ps

[Koegel Buford]
Koegel Buford, J. F. 1994. *Multimedia System,* ACM Press, New York. 451 pages.

[LaLiberte and Braverman]
LaLiberte, D., and Braverman, A. 1995. *A Protocol for scalable group and public annotations.* Computer networks and ISDN systems: Proceedings of

the Third International World-Wide Web Conference, volume 27, number 6, Elsevier. Pages 911-918

[Little and Venkatesh]
Little, T. D. C., and Venkatesh, D. 1994. *Prospects for Interactive Video on Demand.* IEEE Multimedia, volume 1, number 3. Pages 14-24.

[Luotonen]
Luotonen, A., and Altis, K. 1994. *World-Wide Web Proxies.* Computer Networks and ISDN systems, First International Conference on the World-Wide Web, Elsevier Science BV. http://www1.cern.ch/PapersWWW94/luotonen.ps

[Macedonia and Brutzman]
Macedonia, M. R., and Brutzman D. P. 1994. *MBone Provides Audio and Video Across the Internet.* Computer, number 2, April 1994. Pages 30-36.

[McGrath]
McGrath, R. E. 1995. *Performance of Several HTTPD Demons on an HP 735 Workstation.* National Center for Supercomputing Applications, April 25, 1995, online documentation. http://www.ncsa.uiuc.edu/InformationServers/Performance/V1.4/report.html

[Mockapetris87a]
Mockapetris, P. 1987. *Domain Names - Concepts and Facilities.* Internet Request for Comments RFC 1034, November, 1987. 55 pages.

[Mockapetris87b]
Mockapetris, P. 1987. *Domain Names - Implementation and Specification.* Internet Request for Comments RFC 1035, November, 1987. 55 pages.

[Nielsen]
Nielsen, J. 1993. *Usability Engineering*, San Diego, Academic Press. 358 pages.

[Padmanabhan and Mogul]
Padmanabhan, V.N., and Mogul, J.C. 1994. *Improving HTTP Latency.* The Second International WWW Conference '94, Chicago 17-20 October 1994, Mosaic and the Web, Advanced Proceedings, volume 2. Pages 995-1005.

[Partridge et al.]
Partridge, C., Mendez, T., and Milliken, W. 1993. *Host Anycasting Service*, Internet Request for Comments RFC 1546, November, 1993. 9 pages.

[Pitkow and Recker]
Pitkow, J. E, and Recker, M. M. 1994. *A Simple Yet Robust Caching Algorithm Based on Dynamic Access Patterns.* The Second International WWW Conference '94, Chicago 17-20 October 1994, Mosaic and the Web, Advanced Proceedings, volume 2. Pages 1039-46.

[Postel81a]
Postel, J. B. 1981. *Internet Control Message Protocol.* Internet Request for Comments RFC 792, September, 1981. 21 pages.

[Postel81b]
Postel, J. B. 1981. *Transmission Control Protocol.* Internet Request for Comments RFC 793, September, 1981. 85 pages.

[Postel and Reynolds]
Postel, J. B., and Reynolds, J. 1985. *File Transfer Protocol (FTP).* Internet Request for Comments RFC 959, October 1985. 69 pages.

[Ramanathan and Rangan]
Ramanathan, S., and Rangan P. V. 1994. *Architectures for Personalized Multimedia.* IEEE Multimedia, volume 1, number 1. Pages 37-46.

[Reynolds]
Reynolds, F. *A Scalable, Fault Tolerant Web Server.* OSF Research Institute slides, April 21, 1995.

[Sedayao]
Sedayao, J. 1994. *"Mosaic Will Kill My Network!" Studying Network Traffic Patterns of Mosaic Use.* The Second International WWW Conference '94, Chicago 17-20 October 1994, Mosaic and the Web, Advanced Proceedings, volume 2. Pages 1029-1038.

[Smith]
Smith, N. 1994. *What can archives offer the World Wide Web.* Computer Networks and ISDN systems, First International Conference on the World-Wide Web, Elsevier Science BV, 1994. http://www1.cern.ch/PapersWWW94/ngs.ps

[Spero94a]
Spero, S. E. 1994. *Analysis of HTTP Performance problems.* http://sunsite.unc.edu/mdma-release/http-prob.html

[Spero94b]
Spero, S. E. 1994. *Progress on HTTP-NG.* http://www.w3.org/hypertext/WWW/Protocols/HTTP-NG/http-ng-status.html

[Spero95]
Spero, S. E. *Next Generation Hypertext Transport Protocol*, March 26, 1995. http://sunsite.unc.edu/ses/ng-notes.txt

[Trent and Sake]
Trent, G., and Sake, M. 1995. *WebSTONE: The First Generation in HTTP Server Benchmarking*, February 1995. http://www.sgi.com/Products/WebFORCE/WebStone/paper.html

[Turpeinen]
Turpeinen, M. 1995. *Agent-mediated personalized multimedia services*, Master's thesis, Helsinki University of Technology. 86 pages.

[Wessels]
Wessels, D. 1995. *Intelligent Caching for World-Wide Web Objects*, Master's thesis, University of Colorado. 85 pages.

[Worrell]
Worrell, K. J. 1994. *Invalidation in Large Scale Network Object Caches*, Master's thesis, University of Colorado. 49 pages.

[WWW]
World-Wide Web serversoftware.
http://www.w3.org/hypertext/WWW/Servers.html

[Xerox]
Xerox Corporation. *Network Binding Protocol.* Technical report, Xerox Corporation, Network System Institute, Palo Alto, California, June 1986.